Analytical Investigation of On-Path Caching Performance in Information Centric Networks

A PhD Thesis Submitted to the
College of Graduate and Postdoctoral Studies
in Partial Fulfillment of the Requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Alireza Montazeri

©Alireza Montazeri, August/2018. All rights reserved.

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
University of Saskatchewan
176 Thorvaldson Building, 110 Science Place
Saskatoon, Saskatchewan, S7N 5C9
Canada

Dean
College of Graduate and Postdoctoral Studies
University of Saskatchewan
116 Thorvaldson Building, 110 Science Place
Saskatoon, Saskatchewan, S7N 5C9
Canada

Abstract

Information Centric Networking (ICN) architectures are proposed as a solution to address the shift from host-centric model toward an information centric model in the Internet. In these architectures, routing nodes have caching functionality that can influence the network traffic and communication quality since the data items can be sent from nodes far closer to the requesting users. Therefore, realizing effective caching networks becomes important to grasp the cache characteristics of each node and to manage system resources, taking into account networking metrics (e.g., higher hit ratio) as well as user's metrics (e.g. shorter delay). This thesis studies the methodologies for improving the performance of cache management in ICNs. As individual sub-problems, this thesis investigates the LRU-2 and 2-LRU algorithms, geographical locality in distribution of users' requests and efficient caching in ICNs.

As the first contribution of this thesis, a mathematical model to approximate the behaviour of the LRU-2 algorithm is proposed. Then, 2-LRU and LRU-2 cache replacement algorithms are analyzed. The 2-LRU caching strategy has been shown to outperform LRU. The main idea behind 2-LRU and LRU-2 is considering both frequency (i.e. metric used in LFU) and recency (i.e. metric used in LRU) together for cache replacement process. The simulation as well as numeric results show that the proposed LRU-2 model precisely approximates the miss rate for LRU-2 algorithm.

Next, the influence of geographical locality in users' requests on the performance of network of caches is investigated. Geographically localized and global request patterns have both been observed to possess Zipf (i.e. a power-law distribution in which few data items have high request frequencies while most of data items have low request frequencies) properties, although the local distributions are poorly correlated with the global distribution. This suggests that several independent Zipf distributions combine to form an emergent Zipf distribution in real client request scenarios. An algorithm is proposed that can generate realistic synthetic traffic to regional caches that possesses Zipf properties as well as produces a global Zipf distribution. The simulation results show that the caching performance could have different behaviour based on what distribution the users' requests follow.

Finally, the efficiency of cache replacement and replication algorithms in ICNs are studied since ICN literature still lacks an empirical and analytical deep understanding of benefits brought by in-network caching. An analytical model is proposed that optimally distributes a total cache budget among the nodes of ICN networks for LRU cache replacement and LCE cache replication algorithms. The results will show how much user-centric and system-centric benefits could be gained through the in-network caching compared to the benefits obtained through caching facilities provided only at the edge of the network.

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisor, Prof. Dwight Makaroff, for his support, motivation, enthusiasm, guidance and for providing many helpful and beneficial revisions to the written materials throughout my Ph.D. study.

Besides my supervisor, I would like to thank the committee members, Prof. Derek Eager, Prof. Ian Mcquillan and Dr. Aryan Saadat Mehr as well as the external examiner, Prof. Mohamed Hefeeda, for their insightful comments.

I would also like to acknowledge the University of Saskatchewan for supporting me financially. Part of the thesis was funded by the Natural Sciences and Engineering Research Council (NSERC) of Canada through NSERC Engage Grant and NSERC Engage Plus Grant awarded to Prof. Dwight Makaroff.

I would also like to thank Nicholas Beaton from the Department of Math and Stats that collaborated in solving the derivation and verification of the LRU-2 model in Chapter 4.

Last but not the least, I would like to thank my parents who have been the back bone in my personal and academic life for their supports, source of love, strength and encouragement through these years. I am especially thankful to my wife who strengthened me in difficult situations. I also thank my lovely sisters for being supportive throughout my academic life.

Contents

List of Tables

List of Figures

List of Abbreviations

ARC	Adaptive Replacement Cache
CCN	Content Centric Networking
CDN	Content Distribution Network
CS	Content Store
DHT	Distributed Hash Tables
DONA	Data-Oriented Network Architecture
FIB	Forwarding Information Base
FIFO	First In First Out
ICN	Information Centric Networking
i.i.d	independent and identically distributed
IRM	Independent Reference Model
IP	Internet Protocol
ISP	Internet Service Provider
LCD	Leave Copy Down
LCE	Leave Copy Everywhere
LFU	Least Frequently Used
LRU	Least Recently Used
MCD	Move Copy Down
NDN	Named Data Networking
NRR	Nearest Replica Routing
P2P	Peer-to-Peer
PASTA	Poisson Arrivals See Time Averages
PIT	Pending Interest Table
PLST	Pending Local Search Table
SPR	Shortest Path Routing
TTL	Time To Live
VoD	Video on Demand

Chapter 1

Introduction

Analysis of Internet traffic has shown that a high percentage of the Internet traffic is caused by video streaming websites (e.g. YouTube) and Peer-to-Peer (P2P) networks which means Internet users are looking for videos and sharable content (e.g. photo and music) [25]. In other words, the trend of today users' requests conflicts with the current host-centric Internet architecture in which users need to connect to the hosts to get data items of their interest. Literally, users are looking for content and not hosts. This conflict has researchers thinking about a new Internet architecture which adapts to the users' request trends [6]. This new content-centric architecture has been called Information Centric Networking (ICN).

TRIAD was the first ICN network proposed by Cheriton *et al.* [27], which proposed name-based information communication. Since then, researchers have proposed various architectures for ICNs. The Data-Oriented Network Architecture (DONA) [55] project was proposed in 2006 at UC Berkeley. The main contribution of DONA was focused on improving the security and architecture of TRIAD. The Publish Subscribe Internet Technology (PURSUIT) [41] project, a continuation of the Publish Subscribe Internet Routing Paradigm (PSIRP) [35] project, both funded by the EU Framework 7 Program (FP7), have proposed a publish/subscribe protocol stack that replaces the IP protocol stack. Another project supported by European FP7 4WARD project is Network of Information (NetInf) [31], which was continued under the name of Scalable and Adaptive Internet Solutions (SAIL) project.[1] Similarly, Van Jacobson *et al.* proposed the Content Centric Networking (CCN) project [49] in 2009. The main target of all the aforementioned approaches is to improve the performance and end-user experience in the Internet by providing access to content and services by name rather than by original location. This is achieved by changing the concept of host-to-host architecture into host-to-content architecture and by exploiting in-network storage of content.

The content dissemination mechanism in ICNs includes the following processes:

1. **Content Advertisement**: Content publishers advertise newly released content. The system is responsible to create the routes from the edge of the network to the publishers. Edge nodes are the nodes that are connected directly to the users. Intermediate nodes are the nodes that handle the communication between edge nodes and the publishers. In this thesis, content publisher, server and source node are used interchangeably. The content provider for data item i also refers to either content publisher or an ICN node that has a copy of data item i.

[1] //www.sail-project.eu/

2. **Content Discovery/Delivery**: Figure 1.1 shows the process of content discovery/delivery in ICNs. For each content publisher, ICN constructs an overlay tree consisting of all ICN nodes rooted at the ICN node that is closest to the content publisher. ICN constructs one such tree for each content publisher. This tree is used for the content discovery/delivery mechanism. Each node is equipped with three tables: 1) Forwarding Information Base (FIB), 2) Pending Interest Table (PID) and 3) Content Store (CS). Figure 1.2 depicts the changes in these tables for the process of content discovery/delivery. Assume Subscriber1 is intereted in data item i that is published by publisher2. Subscriber1 requests for i through forwarding an Interest message for i (phase 1 in Figure 1.1). A node receiving the Interest forwards it based on its FIB (Phases 2 and 3 in Figure 1.1). The overlay trees are used to fill the FIBs. The FIB of node b0 for example, determines the next ICN node on the path from b0 to the destination of an Interest message (Publisher2). The Interest forwarded by a node is added to the node's PIT (Figures 1.2a and 1.2b); thus, the node can remember the interface on which the corresponding data for the Interest should be sent back (Figure 1.2a and 1.2b). When the Interest arrives at Publisher2, a copy of data item i is sent to the network (phase 4 in Figure 1.1). Any node receiving the data on the path from the content publisher to the user, like $b0$ and $b1$, may cache the data. These nodes then use their PIT to forward the data to the interested users for that data (Phases 5 and 6 in Figure 1.1). The table entry in these nodes' PIT that corresponds to the data will be removed as well as shown in Figures 1.2c and 1.2d.

3. **Caching**: In the current Internet architecture, the intermediate nodes have no idea about the content of the data item they are forwarding. In fact, they are only aware of connections between hosts in the Internet. However, the nodes in ICNs are aware of the content they forward. Equipping ICN nodes with cache storages results in a network of caches. The goal of this network of caches is moving the most popular data items to the network edges. Tewari *et al.* [88] declare that firstly, a caching strategy should minimize the distance to access data (shorter response time). Secondly, a caching strategy should share data items among many users (higher hit ratio). If the data items are cached on the delivery path, it is called *on-path* caching, otherwise *off-path* caching (explained in details in Section 2.5). The ICN in Figure 1.1 for example, deploys on-path caching. Caching in this figure can help the subscribers access their data items of interest quicker. For instance, suppose Subscriber1 has recently requested data item i and the data item is delivered to it. Assume also that data item i has been cached at b0 on the path from Publisher2 to Subscriber1. If Subscriber2 now asks for data item i, its Interest message gets forwarded to b3 first and then b0. If b0 still has i cached in its CS, a copy of i is sent back to subscriber2 that is 2 hops away from Subscriber2. Notice that if no ICN node has a copy of i cached in its CS, the Interest message for i from Subscriber2 has to go through the network to get to Publisher2 that is 4 hops away from Subscriber2.

Storing data items temporarily in a location knowing that the data items are accessed frequently or in a closely related future is defined as caching. In the context of Operating Systems for example, the CPU uses a

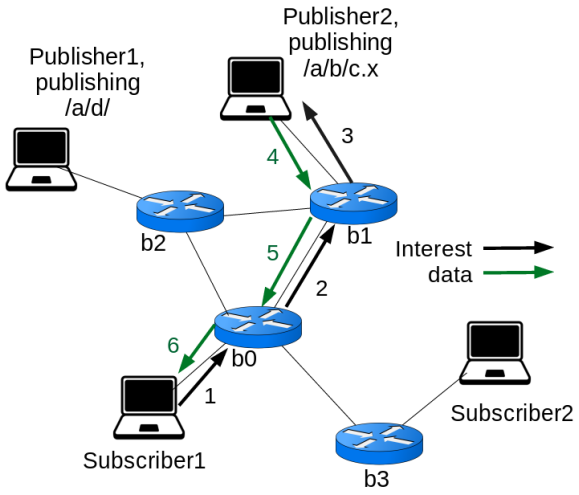


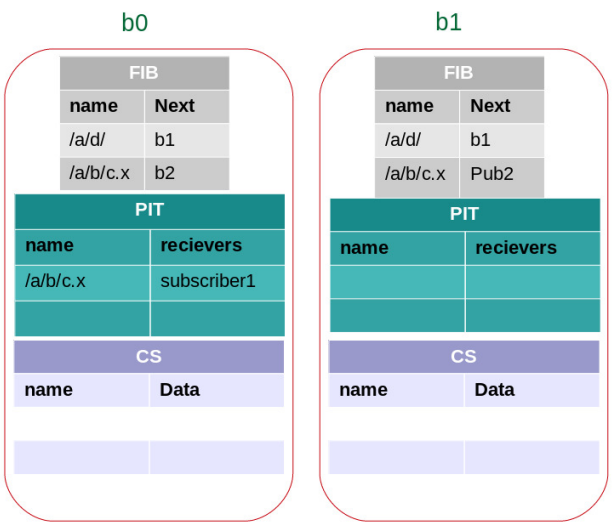
Figure 1.1: Content discovery/delivery in ICNs.

hierarchy of caches to store data. L1 cache places at the top of this hierarchy that has the smallest capacity and highest speed and is closest cache to the CPU. L2, L3 and L4 are placed at lower places of this hierarchy. The memories at each level have larger capacity and slower speed compared to caches at their higher levels. This hierarchy of caches stores the data that the CPU is most likely to require next. Because of limited size of these caches, all the data cannot be cached in this hierarchy. Therefore, the CPU uses the following two algorithms:

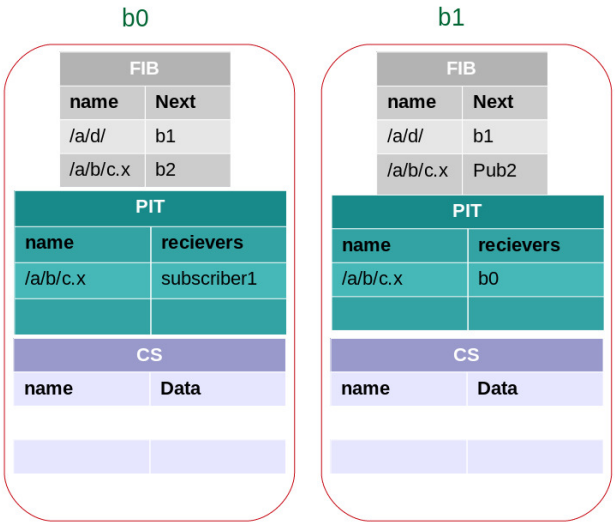
- Cache Replication Algorithm: this algorithm decides what piece of data is worth getting cached (explained in details in Section 2.5).
- Cache Replacement Algorithm (eviction policy): this algorithm decides what piece of data should be evicted from the cache when the cache gets full (explained in details in Section 2.2).

The goal of the simultaneous deployment of these two algorithms is to ensure that the next bit of data that CPU needs is already loaded into L1, resulting in a cache hit. If the data that CPU needs is not found in L1 (a cache miss), the CPU looks for the data in L2. If the data item is not found in L2, the CPU searches L3 and so on. Note that, looking at more caches is taking more time that ends in longer process time of finding a piece of data. A good combination of two algorithms keeps the data items that the CPU will need in a closely related future as close as possible to the CPU (e.g. L1 and L2), resulting in a shorter average access time to data.

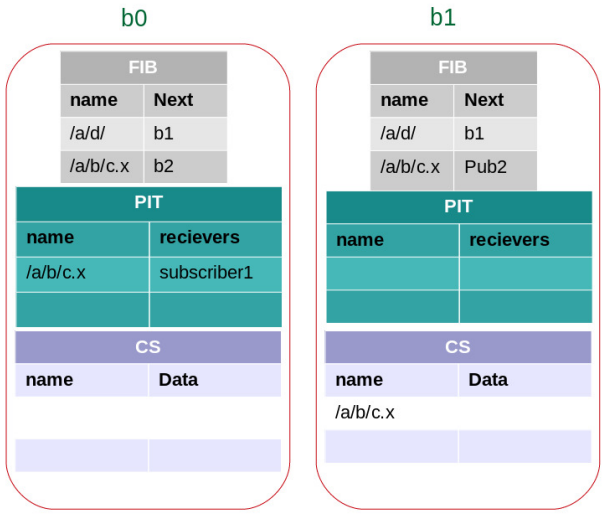
In the context of the Internet, Content Distribution Networking (CDN) and P2P networks deploy caching storages at the edge of the network to reduce the high cost of streaming of data items as well as access time to data item from users' point of view. Contrary to CDN and P2P networks, ICN equips not only the ICN nodes at the edge of the network but also the intermediate nodes. Several studies have been conducted to



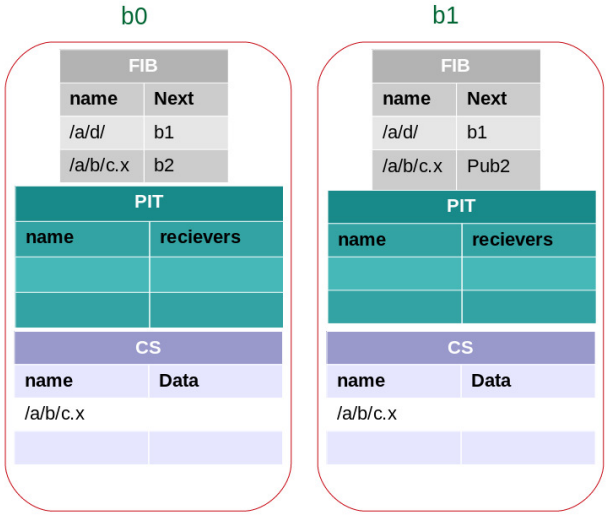
(a) Tables after step 1 in Figure 1.1



(b) Tables after step 2 in Figure 1.1



(c) Tables after step 4 in Figure 1.1



(d) Tables after step 5 in Figure 1.1

Figure 1.2: Changes of FIB, PIT and CS in the process of content discovery/delivery in ICNs

resolve the issues of caching in the web as it can be related to ICN. Despite all the contributions in ICN architectures, standardization of in-network caching is still lacking [92, 98].

1.1 Research Goals

The following three issues are addressed in this thesis:

- **Modelling Cache Replacement Algorithms**: Having an accurate model to calculate the miss rates of different replacement algorithms is crucial for performance analysis of large-scale interconnected caches deployed in ICNs. Evaluating the performance of cache networks is challenging as Dan *et al.* argue that the computational cost to approximate the behaviour of a single Least Recently Used (LRU) grows exponentially as a function of cache size and the number of data items [30]. Besides the studies that modelled different cache replacement algorithms, there is no study that modelled LRU-k [68]. LRU-k is the first algorithm that considers both recency and frequency in its eviction policy (explained in details in Section 2.2), that results in a higher hit ratio compared to LRU. High run-time complexity of this algorithm however, contributed in shaping other cache replacement algorithms that use frequency and recency in their eviction policies with a low run-time complexity.

- **Generating Realistic Users Requests**: Studying the performance of different caching algorithms in ICN is impossible if there is not a good understanding of user request distributions in such networks. Most studies use simplifying assumptions for user request patterns, such as identical request patterns in absence of any trace-driven methodologies since ICNs are not yet deployed. Requests for Internet services have been modelled as Zipf distribution for many years [2, 36]. In a Zipf distribution, a small fraction of all data items have high request frequencies while a large fraction of data items have low request frequencies. Geographically localized request patterns at the edge of the network (e.g. number of views for YouTube videos on a university campus) and global request patterns (e.g. total number of views of YouTube videos) have both been observed to possess Zipf properties [47, 82], although the local distributions are poorly correlated with the global distribution [100]. This suggests that several independent Zipf distributions combine to form an emergent Zipf distribution in real client request scenarios. Having an algorithm that divides a global Zipf distribution into several sub-distributions with Zipf properties in order to create more realistic traffic, consisting of requests that have geographic locality, and studying the influence of output of this algorithm on the caching performance of ICNs for various topologies indicates how misleading the findings based on non-realistic traffic can be in regards to the caching performance of ICNs.

- **Optimal Cache Budget Distribution For Hierarchical Trees of Caches**: Caching facilities in on-path caching can be deployed by all (e.g. CCN [49]) or some (e.g. centrality-based caching [18]) of the ICN nodes on the path of delivering data items from the source to the users. However, some

inconsistent conclusions are implied from different studies about the efficiency of in-network caching. For example, Danzig *et al.* [32] and Rossini *et al.* [79] believe that in-network caching can be more effective. Fayazbakhsh *et al.* [37] and Psaras *et al.* [72] on the other hand believe caching closer to the network edge is more effective. This suggests that ICN literature still lacks an empirical and analytical deep understanding of benefits brought by in-network caching. To find out if the in-network caching is effective to decrease the overall miss ratio and delay in ICNs, an optimal distribution of a fixed cache budget among all the ICN nodes reveals what fraction of the cache budget can be allocated to the intermediate nodes in order to minimize the miss ratio and delay.

1.2 Contributions

The contributions of this work could be expressed as follows:

- **Modelling LRU-2 Cache Replacement Algorithm**: Garetto's recently proposed 2-LRU caching strategy [44] has been shown to outperform LRU. The main idea behind 2-LRU considers both frequency (i.e. metric used in LFU) and recency (i.e. metric used in LRU) together for cache replacement process. This idea was deployed for the first time in LRU-2 algorithm introduced by O'Neil *et al.* in 1993 [68]. The two algorithms however, have different implementations for considering both frequency and recency it their eviction policies. The 2-LRU algorithm uses 2 LRU caches while the LRU-2 algorithm uses a sorted list of the history of requests for data items. As a result, LRU-2 has a high run-time complexity compared to 2-LRU which makes the deployment of LRU-2 challenging at ICN nodes. The two algorithms are explained in Section 2.2. A mathematical model for LRU-2 algorithm is proposed in this work for the first time. The accuracy of the model is studied. The LRU-2 and 2-LRU algorithms are also compared using simulations and mathematical models for individual caches as well as hierarchical trees of caches.

- **Generating more Realistic Synthetic Traffic and Providing Local Search for Content Discovery/Delivery Mechanism**: An algorithm that can generate realistic synthetic traffic is proposed. The algorithm generates requests for users of local regions such that the distribution of users' requests in each local region possesses Zipf properties. The algorithm also guarantees that combination of users' requests in the local regions forms a Zipf distribution. The simulation results show that the caching performance could have different behaviour based on what distribution the users' requests follow. In addition to the standard content discovery mechanism, explained earlier in this chapter, a local search for content discovery/delivery mechanism is proposed. The goal of this mechanism is to help the ICN nodes at the edge of the Internet discover requested data items in their neighbouring ICN nodes before they send the Interest messages toward the source node. The simulation results show that the proposed local search improves all the performance metrics.

- **Optimal Cache Budget Distribution For Hierarchical Trees of Caches**: To investigate the benefits of in-network caching, an analytical model is proposed that optimally distributes a total cache budget of C among the nodes of ICN networks under non-IRM environment. The cache budget distribution is studied regarding optimizing the following metrics: 1) system-centric metrics (e.g. total hit ratio in ICNs), user-centric metrics (e.g. average time to access data), and 3) a combination of system-centric and user-centric metrics. The findings of this contribution reveal the efficiency of in-network caching as well as the optimal distribution of cache budget in ICNs.

1.3 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 reviews prior works on modelling the caching systems in the context of ICN. The experimental environment is briefly discussed in Chapter 3. Chapter 4 proposes a mathematical model for LRU-2 algorithm. A comparison between 2-LRU and LRU-2 algorithms is performed in this chapter as well. Chapter 5 develops and evaluates an algorithm that can generate realistic synthetic traffic to regional caches that possesses Zipf properties as well as produces a global Zipf distribution. Chapter 6 models the cache distribution in a network of caches as an optimization problem. The solution for this optimization problem reveals the efficiency of in-network caching. Chapter 7 summarizes the thesis and outlines possible future works.

Chapter 2

Background and Related Work

Caching facilities are critical to achieving ICN's advantages such as enhancing the delivery of information (reduced delay) and reducing the overall load on source nodes. This thesis takes into account *frequency*, *recency*, *cost of retrieval*, *replication* and *replacement* as the factors influencing the cache behaviour. Frequency shows how frequently a data item is requested. Recency corresponds to the relative times that a data item is requested. Cost of retrieval means the cost to access a data item. Replication means what nodes in a network of caches can cache a copy of a given data item. Finally replacement determines what data items should be evicted when the cache is full.

In this chapter, a short description of Zipf distributions is given in Section 2.1. Cache replacement algorithms are reviewed in Section 2.2. The works proposed for modelling individual caching algorithms under Independent Reference Model (IRM) are covered in Section 2.3 in order to find a model that is expandable to different cache replacement algorithms including 2-LRU. IRM however, ignores the dynamics of data items' popularity such as temporal and geographical locality. Consequently, models that approximate the individual cache behaviour under IRM should be extended under non-IRM to enable realistic analysis of performance of individual caches as well as network of caches. In this regard, modelling individual caching algorithms under a non-IRM environment, in which temporal and geographical locality is considered in data items' popularity, is covered in Section 2.4. Next, Section 2.5 reviews in-networking caching approaches as well as the models proposed for in-network caching in the context of ICN. Techniques covered in this section can be used to find the optimal distribution of the cache budget among the ICN nodes. Finally, Section 2.6 summarizes the chapter.

2.1 Zipf Distribution

The correlation between the frequency of data items (how often they are requested) and their rank has been shown to follow a Zipf distribution [3, 16, 20, 23, 47, 62]. In this distribution, if data items are ranked according to their frequency where the most frequently requested data item has the first rank, the frequency of requests for data item i with rank r is inversely proportional to its rank; i.e. $p_i \approx r^{-\alpha}$. A rank/frequency distribution with Zipf proportion is long-tailed, depicted in Figure 2.1, in which a small set of data items with high frequency (short head) is followed by a large set of data items with low frequency (long tail) which

gradually *tails off* [4]. The data items at the far end of the distribution have a very small probability of requests. Foss *et al.* give the following mathematical definition for long-tailed functions: positive function $f(x)$ is long-tailed, if and only if:

$$\lim_{x\to\infty} \frac{f(x+y)}{f(x)} = 1, \quad \forall y > 0 \text{ [40]}. \tag{2.1}$$

Using this definition, Zipf function $f(x) = x^{-\alpha}$ is long-tailed since

$$\lim_{x\to\infty} \frac{f(x+y)}{f(x)} = \lim_{x\to\infty} \frac{(x+y)^{-\alpha}}{x^{-\alpha}} = 1, \quad \forall y > 0.$$

Note that in log-log scale, the rank/frequency distribution is transformed to $\log(p_i) \approx -\alpha \log(r)$. In this thesis, the *frequency* and *popularity* of data items are used interchangeably; the more frequently data item i is requested, the more popular data item i is.

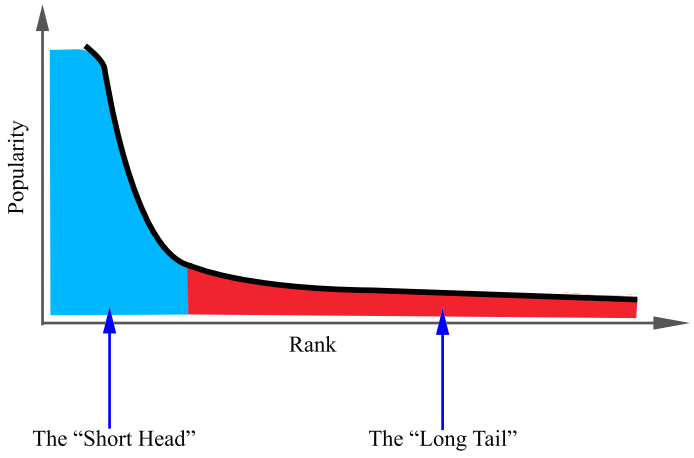


Figure 2.1: Long-tailed distribution.

Researchers have found values of α between 0.56 [47] and 2 [15] for Zipf distributions for different types of data items. Fricker *et al.* studied the popularity of 270000 torrents on Demonoid site [43]. They found Zipf distribution with $\alpha = 0.82$ for the files. Cha *et al.* found a Zipf distribution with $\alpha = 0.8$ for user generated content [16]. A study on YouTube videos by Chowdhury *et al.* find $\alpha \in (1.15, 1.38)$ for different categories of videos, such as news, music, etc. [22].

However, a 100% fit between observations and Zipf distributions does not always occur. Chowdhury *et al.* for example, show that popularity distributions of YouTube music videos follow Zipf distributions while the heavy tail part of the distributions can be fit with a Weibull cutoff [22] (Figure 2.2). Another study by Cha *et al.* found that YouTube science videos fit a Zipf distribution with an exponential cut-off [16]. Despite the last part of the curve that does not fit a Zipf distribution, researchers assume Zipf distributions to study different caching systems [42, 43, 44].

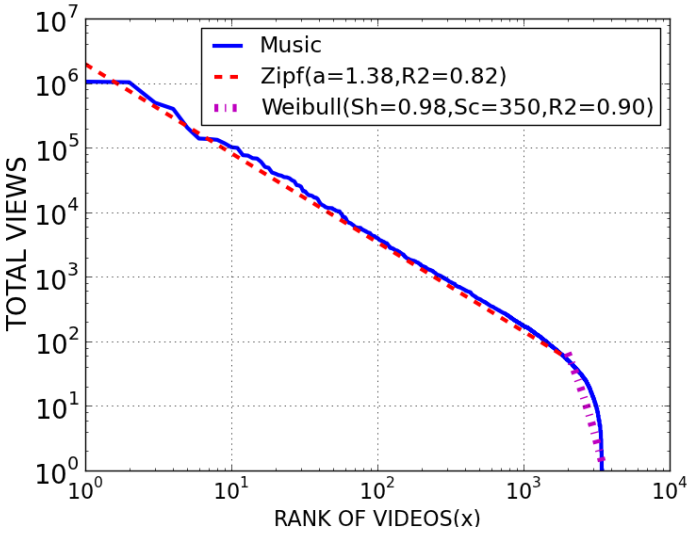


Figure 2.2: Zipf distribution with Weibull cut-off, [22].

2.2 Cache Replacement Algorithms

When a request for a data item arrives at a cache, a cache *hit* occurs if the requested data item is stored in the cache; otherwise it is a cache *miss*. The performance of cache replacement algorithm ξ is often measured through its hit ratio, shown by h_ξ as follows

$$h_\xi = 100 \times \frac{\text{the number of accesses that result in a cache hit}}{\text{the total number of accesses}}.$$

Cache replacement algorithms choose what data items to evict from the cache, when the cache is full, in order to make room for other data items. Cache replacement algorithm ξ_1 outperforms ξ_2 if the hit ratio of ξ_1 is higher than the hit ratio provided by ξ_2 (i.e. $h_{\xi_1} > h_{\xi_2}$). With a cache capacity of C in an IRM environment, the Least Frequently Used (LFU) replacement algorithm stores the C most popular data items in the cache and provides optimal performance under the IRM [26]. In other words, LFU evicts a data item with the lowest frequency based on the history this cache replacement algorithm records about the observed frequency of data items. In a non-IRM environment however, Garetto *et al.* show that LFU is not the optimal cache replacement algorithm when popularity of data items changes over time [44]. When used in practice, it adapts poorly to temporal locality, caching stale items with past high frequency [45].

LRU is another replacement algorithm with low run-time cost of $O(1)$. Upon the arrival of data item i, if data item i is already at the l^{th} spot in the queue, LRU moves i to the head of the queue and other data items, located between the head and $l-1^{st}$ spot, one spot down in the queue. If data item i is not in the queue, LRU pushes i at the head of the queue and moves all other items in the queue one spot down. In this

case, the data item at the tail of the LRU queue is evicted if the queue is full. In other words, LRU keeps the most recent accessed data items closer to the head of the queue. LRU however, caches unpopular data items (i.e. data item with low frequency) that have even a single request (called *one-timers*), potentially evicting popular data items. To deal with this issue, algorithms like q-LRU, LRU-k, k-LRU, LRU-2Q and ARC have been proposed. All these algorithms do not let unpopular data items get inserted into the cache. For q-LRU, while its eviction policy is similar to LRU, a data item is cached at the head of the LRU queue with probability q. The more often a data item is requested, the higher chance it has to get inserted into a q-LRU cache. Garetto *et al.* show that q-LRU tends to LFU as q goes to zero [44]. Consequently, this caching algorithm prevents data items with low request frequency from getting inserted into the cache.

O'Neil *et al.* proposed LRU-k, which modifies LRU to keep track of the last k requests for each data item [68]. They assume $D = \{1, 2, \ldots, i, \ldots, n\}$ as a list of data items, and request arrival for data items specified by a reference string like $r_1, r_2, \ldots, r_t$, where $r_t = i \in D$ means that data item i is requested at time t. O'Neil *et al.* assume for any given instant t, data item i has probability of b_i to be next request arrival at LRU-k; i.e. $P(r_{t+1} = i) = b_i$. This assumption suggests that the reference string is probabilistic. They then define *backward k-distance* as following: given a reference string $r_1, r_2, \ldots, r_t$, the backward k-distance for data item i from time t, shown by $b_t(i,k)$, is the distance backward to the k^{th} most recent reference for data item i:

$$b_t(i,k) = \begin{cases} x & \text{if } k \text{ references for } i \text{ exist in the reference string in the most recent } x \text{ references,} \\ \infty & \text{if } i \text{ appears fewer than } k \text{ times in the entire reference string.} \end{cases}$$

O'Neil's LRU-k algorithm evicts a data item i with largest $b_t(i,k)$; LRU is used if more than one data item with $b_t(i,k) = \infty$ are already cached. In LRU-2 for example, if the reference string from r_1 to r_{13} occurs as following

$$i,\ j,\ l,\ m,\ i,\ i,\ l,\ j,\ n,\ m,\ l,\ i,\ n,$$

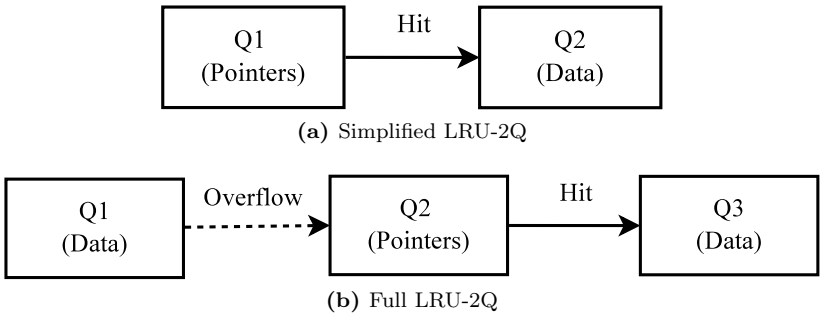
LRU-2 keeps a sorted meta-data list of items based on their 2-backward distance which would be as follows:

$$b_t(j,2) = 11,\ b_t(m,2) = 9,\ b_t(i,2) = 7,\ b_t(l,2) = 6,\ b_t(n,2) = 4.$$

Having capacity of $C = 5$ for this LRU-2 cache, if item h is requested at time 14, LRU-2 evicts item j since $b_t(j,2)$ is the largest among the cached data items in LRU-2. LRU-k results in a higher hit ratio compared to LRU, but its implementation complexity is $O(\log C)$ since LRU-k needs to keep the meta-data list sorted on each request arrival.

This encouraged researchers to design cache replacement algorithms which produce equivalent hit ratios while eliminating implementation overhead. In this regard, $O(1)$ algorithms LRU-2Q [51], k-LRU [44] and Adaptive Replacement Cache (ARC) [63] have been proposed, considering both recency and frequency of requests.

A simplified version of LRU-2Q, depicted in Figure 2.3a, uses a First In First Out (FIFO) cache Q_1 and LRU cache Q_2. When item i is requested for the first time, a reference to i gets cached in Q_1. Cache Q_1,



(a) Simplified LRU-2Q

(b) Full LRU-2Q

Figure 2.3: Simplified/full LRU-2Q cache [51].

which is called a *ghost* cache, remembers the access history of one-timers through references. If i is requested again and its reference is already stored in Q_1, it then gets cached in Q_2. The performance of the simplified LRU-2Q drops for non-stationary traffic.

Therefore, the full LRU-2Q algorithm was proposed. As shown in Figure 2.3b, Full LRU-2Q consists of two FIFO caches: Q_1, Q_2 and a LRU cache Q_3. A data item is cached in Q_1 the first time it is accessed. The oldest data item in Q_1 is pushed out whet this queue overflows. The evicted data item is then stored as a ghost cache entry in Q_2. If a data item is requested and its reference is in Q_2, the data item is inserted at the head of Q_3. If the data item is requested and it is already cached in Q_3, the data item is moved to the head of Q_3.

ARC cache is depicted in Figure 2.4. ARC has two LRU caches called L_1 and L_2; L_1 stores the data items accessed only once and L_2 stores data items requested at least twice. Furthermore, L_1 is divided into T_1 and B_1 queues. The former stores data items and the latter stores ghost caches. The L_2 list is similarly divided into T_2 and B_2. The sizes of the four queues satisfy the following conditions:

$$0 \leq |L_1| + |L_2| \leq 2C, \quad 0 \leq |L_1| \leq C, \quad 0 \leq |L_2| \leq 2C \quad [63].$$

Assuming a data item is already stored in either T_2 or T_1 or remembered in B_1 or B_2, the data item will be moved to the head of the T_2 queue when the data item is requested again.

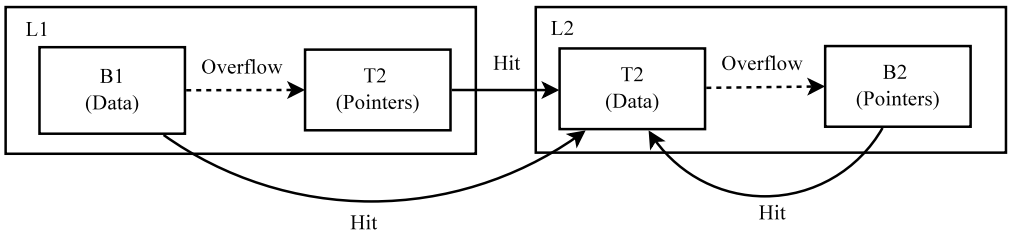


Figure 2.4: ARC cache [63].

Garetto *et al.* proposed k-LRU that consists of same-size $k-1$ virtual LRU caches (i.e. storing the references of data items) and one physical LRU cache (Figure 2.5) [44]. Before arriving at the physical cache (indexed k), requests have to go through $k-1$ virtual caches located in front of it. After a request for a data item at the ICN node arrives, the reference/data item can be stored in cache $l > 1$ only if its reference is already stored in cache $l-1$. In other words, the following two operations are performed when data item i is requested:

- For each LRU cache h in which i is already cached either virtually or physically, data item i is moved to the top of the LRU.
- For LRU cache h in which i is neither stored virtually nor physically, but its reference is stored in $h-1$, reference/data i is then cached at the top of h. The other items stored in h are moved one spot back in h and the item at the end of the cache gets evicted.

The k-LRU algorithm is a generalization of LRU-2Q and ARC. For user requests following a Zipf distribution, Garetto *et al.* show that

- The q-LRU cache with $q = 0.01$ results in a higher hit ratio compared to a q-LRU with $q = 0.1$ as they have shown that this cache replacement algorithm tends to LFU for small q.
- Having a virtual cache before the LRU cache (2-LRU) provides a huge benefit for small caches under IRM.
- LFU performs poorly under non-IRM.
- The 2-LRU cache replacement algorithm provides a good hit ratio under non-IRM. The reason is that 2-LRU considers both frequency and recency. In other words, while 2-LRU prevents the one-timers from getting inserted into the cache, it allows the data items that are getting higher frequency to get inserted into the cache.

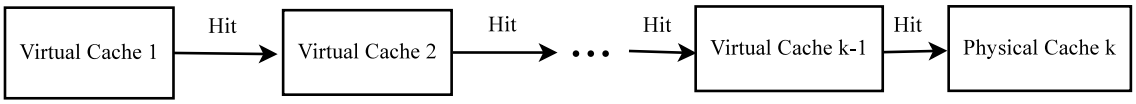


Figure 2.5: k-LRU cache [44].

2.3 Modelling Caching Algorithms Under IRM

Having an accurate model to calculate the miss rates of different replacement algorithms is crucial for performance analysis of large-scale interconnected caches. Evaluating the performance of cache networks is challenging as Dan *et al.* argue that the computational cost to approximate the behaviour of a single LRU

grows exponentially as a function of cache size and the number of data items [30]. However, several models have been proposed [15, 19, 30, 42, 72, 76] that can approximate cache performance at an affordable computational cost under IRM. The IRM is the standard mechanism to characterize the pattern of requests for data items that arrive at a cache. The IRM considers the following two assumptions [44]: (i) there is a fixed number of data items (no new data item is introduced to the catalogue) and (ii) the probability p_i, as the frequency of requests for i, $1 \leq i \leq N$, is constant over time and independent of all past requests. The latter implies that IRM generates an independent and identically distributed (i.i.d.) sequence of requests.

Assuming C and N are the size of cache and the number of data items respectively, Dan and Towsley proposed an approximate technique with complexity $O(CN)$ for the estimation of the hit probability for LRU cache under IRM [30]. Their approach approximates the probability that data item i is in r^{th} position in LRU cache, $P_i(r)$, as

$$P_i(r) = \frac{\big(p_i(1 - b_i(r-1))\big)^+}{\sum_{j=1}^{N} \big(p_j(1 - b_j(r-1))\big)^+} \quad [30],$$

in which, p_i is the popularity ratio of data item i, $(a)^+$ takes value a whenever $a > 0$ and value 0 otherwise; and $b_i(r)$ is the probability that data item i is at any of the first r positions of the LRU cache such that

$$b_i(r) = \sum_{s=1}^{r} P_i(s) \quad [30].$$

Using these two equations recursively, $P_i(r)$ for $1 \leq r \leq C$ can be calculated.

Psaras *et al.* propose a Markovian approach to approximate the hit probability in LRU caches under IRM [72]. Assuming that requests for a data item arrive at an ICN node as a Poisson process with rate of λ and C as the cache capacity of the ICN node, each request for other data items moves this data item to the bottom slot in the node's cache. Assuming that requests which will move the data item further down the cache also arrive as a Poisson process with rate μ, Figure 2.6 shows the continuous time homogeneous Markov chain that models the rank of the data item in the cache [72]. Note that requests for data items that either are not in the cache or are in cache but further down than the data item i can move data item i one slot down to the tail of the cache. The balance equations can be solved to calculate the probability of steady state as follows:

$$P_r = \frac{\lambda}{\mu}\big[\frac{\mu}{\mu + \lambda}\big]^i \quad 1 \leq r \leq C,$$

$$P_{C+1} = \big[\frac{\mu}{\mu + \lambda}\big]^C \quad [72].$$

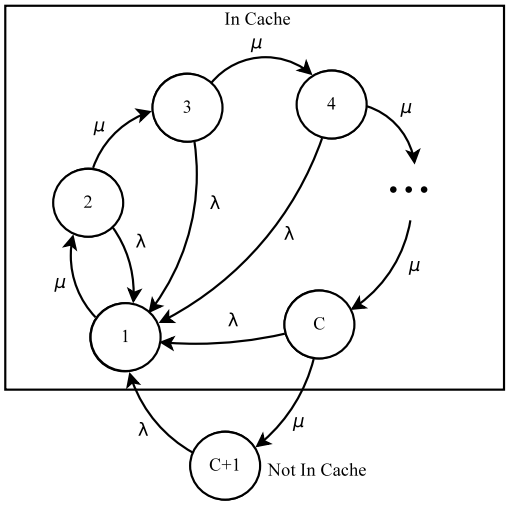


Figure 2.6: Arrival processes of a data item at LRU cache [72].

The probability P_{C+1} corresponds to the miss probability of the data item in Psaras's model for LRU. Having a good approximation of μ in Psaras's model is however challenging. Assuming all requests for other data items arrive at rate ν, all 100% fraction of ν does not compose μ. In this regard, Psaras *et al.* divide ν into ν_c and ν_n as the request rates for data items in the cache and not in the cache respectively, such that $\nu = \nu_c + \nu_n$ [72]. Arrivals in ν_n always move the data item i one spot down in the LRU cache. The arrivals in ν_c however, move i one spot down if their corresponding data items are at further down the cache positions compared to the position of item i. Then, they assume that data items have equal probability to be found in any position in the cache. This assumption calculates an upper bound for data item i's cache miss rate, although the assumption is not realistic. As a result of splitting ν, the rate of moving item i from position j to $j+1$ is given by $\mu = \nu_n + \nu_c(C+1-j)/C$, where the second part represents the arrival of requests for data items that are in the cache, in a position further down than the position of i. Psaras *et al.* find the probability of P_{C+1} in Markov Chain in Figure 2.6 in steady state as follows:

$$P_{C+1} = \Big[\frac{\nu_c + \nu_n}{\nu_c + \nu_n + \lambda}\Big]^C \Big[1 + O\Big(\frac{\nu_c}{C(\nu_c + \nu_n)}\Big)\Big]\Big[1 + O\Big(\frac{\nu_c}{C(\nu_c + \nu_n + \lambda)}\Big)\Big] \quad [72]. \tag{2.2}$$

For non-equal probability for a data item to be found in different positions in the cache, $\mu = \nu_n + \nu_c(C+1-j)/C$ would always be larger and P_{C+1} is hence smaller than (2.2). As a result, (2.2) is an upper bound for the miss probability of data item i.

Another approximation to calculate the hit probability of data item i in LRU caches under IRM was originally proposed by Che *et al.* [19]. Che *et al.* define τ_i as the *characteristic time approximation* of item i that is the time needed before C distinct data items (not including data item i) are requested. In other words, τ_i is item i's time to eviction. Che's approximation assumes τ_i is independent of i; this property is

confirmed by Fricker *et al.* [42] with a Zipf popularity distribution. Fricker *et al.* show that (1) the coefficient of variation of τ_i vanishes as the cache size grows; (2) $\tau_i \approx \tau_j$ $(i \neq j)$ when the catalogue size is sufficiently large.

Having τ as τ_i independent of i, data item i is in the cache at time t if and only if less than τ has elapsed since the last request for item i. Under a Poisson arrival process assumption, for data item i with rate λ_i, the time-average probability P_i^{in} that data item i is in the cache is given by

$$P_i^{in}(\tau) = 1 - e^{-\lambda_i \tau}. \tag{2.3}$$

Che's approximation calculates τ, assuming

$$\sum_{j=1}^{N} P_j^{in}(\tau) = C. \tag{2.4}$$

with item population N and a cache of C items. As an immediate consequence of the Poisson Arrivals See Time Averages (PASTA) property for Poisson arrivals, P_i^{in} also represents the hit probability P_i^{hit} [44]. In the rest of this chapter, $P_i(\tau)$ is used instead of $P_i^{hit}(\tau)$ and $P_i^{in}(\tau)$, unless stated.

As shown in Figure 2.7, consider item i arriving as a Poisson process with rate λ_i at times $t_1, t_2, t_3 \ldots$; let $u_l = t_l - t_{l-1}$ as the time period between l^{th} and $l-1^{st}$ requests. Assume item i was requested at time $t = 0$, and at that point it entered (or re-entered) the cache. Furthermore, say $n \geq 1$ is the smallest value such that $u_n > \tau$. We wish to know the expectation of the sum $S = u_1 + u_2 + \cdots + u_n$, interpreted as the expected time period between each two consecutive cache misses for item i. This can be computed as follows: we must sample u_1, with expected value $\frac{1}{\lambda_i}$. With probability $e^{-\lambda_i \tau}$ we terminate there, otherwise we sample again. What happens after that is independent of u_1, and so has expected sum $\mathbb{E}[S]$. Hence

$$\mathbb{E}[S] = \frac{1}{\lambda_i} + (1 - e^{-\lambda_i \tau})\mathbb{E}[S]. \tag{2.5}$$

The miss rate expectation for item i (λ_i') is

$$\lambda_i' = \frac{1}{\mathbb{E}[S]} = \lambda_i e^{-\lambda_i \tau}. \tag{2.6}$$

This approximation, recognized to be very accurate [42, 44], has since been used to model other caching algorithms. Fricker *et al.* calculate P_i in FIFO cache replacement algorithm using Che's approximation as follows [42]:

$$P_i(\tau) = \frac{\lambda_i \tau}{\sum_{j \neq i}^{N} \lambda_j + \lambda_i \tau} \quad [42].$$

Imai uses Che's approximation to model LRU-2Q and ARC [48]. Garetto *et al.* also use Che's technique to model q-LRU and k-LRU [44]. For q-LRU, a request for data item i is a hit at time t if the previous request has been arrived in $[t - \tau_C, t]$ and either the data item was already in the cache or data item i was inserted into the cache with probability q. Therefore the hit probability of data item i in this cache replacement algorithm is

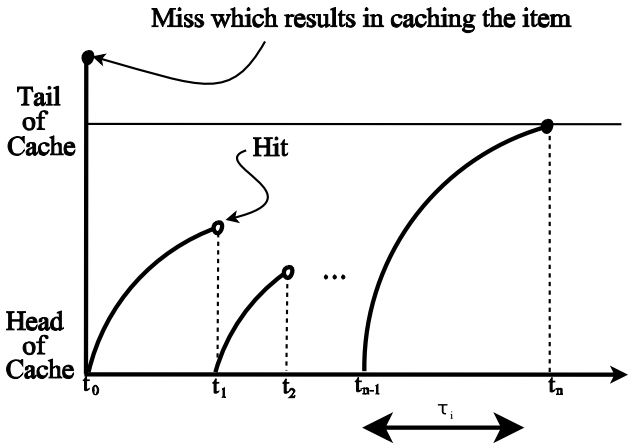


Figure 2.7: Arrival processes of a data item at LRU cache [19].

$$P_i^{hit}(\tau) = (1 - e^{-\lambda_i \tau})\big[P_i^{in}(\tau) + q(1 - P_i^{in}(\tau))\big], \tag{2.7}$$

which can be simplified after applying PASTA to be

$$P_i(\tau) = \frac{q(1 - e^{-\lambda_i \tau})}{e^{-\lambda_i \tau} + q(1 - e^{-\lambda_i \tau})} \quad [44]. \tag{2.8}$$

For k-LRU, a request for data item i at time t is a hit at cache j if the previous request arrived in $[t-\tau_j, t]$ and either the data item was already in cache j or it was not in cache j but its reference was in virtual cache $j-1$. Therefore, the hit probability of data item i is calculated as

$$P_{j,i}^{hit}(\tau_{j-1}, \tau_j) = (1 - e^{-\lambda_i \tau_j})\big[P_{j,i}^{hit}(\tau_j) + P_{j-1,i}^{hit}(\tau_{j-1})(1 - P_{j,i}^{hit}(\tau_j))\big] \quad 1 \le j \le k \quad [44], \tag{2.9}$$

in which τ_j is the time to eviction for j^{th} LRU cache and $P_{j,i}$ is the hit probability of item i at j^{th} LRU. Garetto's model for k-LRU however, assumes the arrival requests at cache $k \ge 2$ follows a Poisson distribution. In fact, the arrivals at cache $k \ge 2$ should be modeled as an ON-OFF Markov modulated Poisson process [46]. As explained in Section 2.2, the reference/data item is stored in cache $l > 1$ only if its reference is already stored in cache $l-1$. Assuming arrivals of requests for data item i at cache 1 follow a Poisson distribution, the requests for item i arriving at cache 2 will not be Poisson. In other words, requests for item i arrive at cache 2 if item i has been already stored in cache 1 (ON phase), and no requests for item i arrive at cache 2 if the item is not cached in cache 1 (OFF phase). Therefore, Gast *et al.* use a Markov chain to model the arrival rates at the caches $k \ge 2$ that results in a more accurate approximation of cache hit probability for k-LRU as follows:

$$P_{j,i}(\tau_1, \tau_2, \ldots, \tau_j) = \frac{\Pi_{s=1}^{j}(1 - e^{-\lambda_i \tau_s})}{\Pi_{s=1}^{j}(1 - e^{-\lambda_i \tau_s}) + e^{-\lambda_i \tau_j}\left(1 + \sum_{l=1}^{j-1} \Pi_{s=1}^{l}(1 - e^{-\lambda_i \tau_s})\right)} \quad 1 \leq j \leq k \quad [46]. \qquad (2.10)$$

2.4 Modelling Caching Algorithms Under non-IRM

Although the IRM assumption is convenient, it is too simplistic in the context of cache networks, where requests for data items exhibit strong correlation in both space and time domains. The IRM ignores the temporal correlations in the sequence of requests, as observed in studies of Internet traffic [1, 12, 17, 84].

The popularity growth of data items is modelled in some studies. Pinto *et al.* [69] and Szabo *et al.* [87] propose models to predict the popularity evolution of data items using historical information provided by early popularity measures. Szabo *et al.* for example, studied two dataset collected from YouTube and Digg.[1] They propose a model that can predict the popularity growth of data items in YouTube and Digg using the initial popularity of data items and a log-linear distribution for the popularity growth of data items. For example, given the initial popularity of data items in Digg in the first two hours, their model forecasts the popularity of the items 30 days ahead with a relative error of 10%. For YouTube videos, however, the model needs the initial popularity of videos in the first 10 days to achieve the same relative error.

Borghol *et al.* collect a data set of user generated videos on YouTube to study the popularity growth of videos over time [12]. Using this dataset, they propose a model to create popularity dynamics. Their model gets some inputs including three distributions for weekly views for videos in the following three phases: before-peak, at-peak and after-peak. Borghol *et al.* use a mixture of beta and lognormal distributions for each of these three phases [12].

Wu *et al.* develop a stochastic fluid model that captures video's popularity evolution using the inherent attractiveness of the videos, information spreading process and the user reaction process [97]. These two processes model how the videos are recommended to the users, and how the users' reaction would be. The goal of their model is predicting the popularity growth of the videos that can be used by Internet Service Providers (ISPs) to improve service qualities.

Shen *et al.* propose a framework that uses a reinforced Poisson process to model popularity growth of citations of paper [85]. Similar to findings of Wu *et al.* [97], Shen *et al.* found out that popularity dynamics of citations is a function of three ingredients: intrinsic attractiveness, the users' reaction to popularity growth of citations and the temporal distribution indicating the aging effect in attracting new attention.

Avramova *et al.* propose the following closed form expression as the cumulative number of views for video i over time [5]:

$$I_i(t) = \rho_i \left[1 - \left[1 + \frac{\left(\beta^{-\frac{1}{\delta_i}} - 1\right)(t - \theta_i)}{\eta_i}\right]^{-\delta_i}\right],$$

[1] http://digg.com

in which a fraction of $(1-\beta)$, $0<\beta<1$, of views are accumulated η_i time units after introduction of video i; parameter ρ_i is the total number of views that video i has over its lifetime and θ_i is the time that video i is introduced to the network. If $\beta_i = 0.5$, η_i is the median, meaning η_i is pointing to a time that the video got half of its views. Parameter δ_i determines the shape of video i's popularity growth. Large δ_i converges $I_i(t)$ to an exponential function; this could be interpreted as video i gathers most of the views right after its introduction to the system. On the other hand, $I_i(t)$ converges to a long-tail distribution as δ_i gets closer to 1; meaning the video continues to receive a high number of views over its lifetime after a long period of its introduction to the system.

Traverso *et al.* extended Che's model [19] to take the time-varying popularity of content explicitly into account, and presented an approximation model for a LRU cache under non-stationary traffic conditions [90, 89]. They proposed Shot Noise Model (SNM) to capture the dynamics of data items. SNM characterizes the request process for data item i using a time-inhomogeneous Poisson process whose frequency rate at time t is given by $V_i\beta(t-\theta_i)$, where $\beta(t)$ represents the shape of popularity growth for data item i, V_i is associated to data item i's total number of requests, θ_i is the time that data item i is introduced into the catalogue and $\beta(t)$ is taken to be an arbitrary function having the following features: (1) $\beta(t) \geq 0\ \forall t$ with $\beta(0^+) > 0$; (2) $\beta(t) = 0\ \forall t < 0$; (3) $\beta(t)$ continuous almost everywhere; and (4) $\int_0^\infty \beta(t)dt = 1$. They also assume that new data items are introduced to the network by rate γ. The average life time of data item i (l_i) is then given by

$$l_i = \frac{1}{\int_0^\infty \lambda_i^2(t)dt} \quad [90].$$

Traverso *et al.* use uniform, exponential and power-law distributions as the shape of $\beta_i(t)$. However, they find out that an accurate prediction for the hit probability of data item i in LRU cache is dependent on l_i regardless of the shape of $\beta_i(t)$.

Using SNM and other methods modelling the dynamics of data items' popularity however, is significantly challenging for studying the performances of cache systems. For example, Traverso *et al.* approximate item i's hit probability in LRU to be:

$$P_i = 1 - \int_0^\infty \beta(t) \frac{\phi_V'\Big(-\int_0^{\tau_C} \beta(t-\theta)d\theta\Big)}{E(V)} dt \quad [90].$$

They assume that $V_i (1 \leq i \leq m)$ form an i.i.d. sequence of random variables distributed around some reference V. They also denote $\phi_V(x) = E(e^{xV})$ to be the moment generating function of V and $\phi_V'(x)$ as its first derivative. Characteristic approximation time τ is the solution of the following equation:

$$C = \gamma \int_0^\infty 1 - \phi_V\Big(-\int_0^\tau \beta(t-\theta)d\theta\Big)dt \quad [90]. \tag{2.11}$$

Olmos *et al.* use the findings of Traverso *et al.* (i.e. l_i is enough to have an accurate prediction of cache performances) to define λ_i such that $V_i = \lambda_i l_i$ [67]. They then estimate the average number of hits in a LRU cache as following:

$$\int_{\lambda>0}\int_{l<\tau_C}\left[\lambda l - 1 + e^{-\lambda l}\right] f(\lambda,l)\mathrm{d}\lambda\ \mathrm{d}l \quad + \\ \int_{\lambda>0}\int_{l>\tau_C}\left[(\lambda l - 1)(1-e^{-\lambda\tau_C}) + \lambda\tau_C e^{-\lambda\tau_C}\right] f(\lambda,l)\mathrm{d}\lambda\ \mathrm{d}l \quad [67], \tag{2.12}$$

where, the pair of (λ, l) is distribution of any (λ_i, l_i).

Using SNM for LRU results in complicated estimations (2.11), (2.12). Garetto *et al.* dispute that estimating the performance of other cache replacement algorithms with SNM will be impractical [45]. Using Che's approximation, an explicit approximation for the hit probability of data item i in a LRU cache at time t is expressed as $1 - Pr\{\text{no requests for data item } i \text{ arrive in } [t-\tau, t]\}$. This approximation is computed easily under stationary (homogeneous) and time-varying (inhomogeneous) Poisson processes. However, this approximation for other cache replacement algorithms can only be computed under stationary Poisson arrival processes of content requests. For example, the dynamics of a cache with RANDOM cache replacement algorithm using Che's approximation are reduced to a G/M/1/0 queue. Therefore, the hit probability of a data item in this cache is then equal to the probability of finding the server of this queueing system busy upon the arrival. An explicit expression of this probability can be only calculated under stationary conditions. Under non-stationary (transient) conditions however, the hit probability can be only expressed as a solution of a system of differential equations that makes the computation of hit probability excessively complicated.

Garetto *et al.* show that the performance of caching systems under the SNM traffic model can be accurately approximated by having a fixed-size content catalogue when the arrival process of each data item is modelled by a renewal process with a specific inter-request time distribution [45]. This approach is used to extend the approximation models for predicting the performance of caching algorithms under IRM to non-IRM. Specifically, the request process for data item i in Garetto's work is described by an independent renewal process with assigned inter-request time distribution as follows [44]: assuming $F_i(t)$ as the CDF of the inter-request time t for data item i, the average request rate λ_i for data item i is then given by

$$\lambda_i = \frac{1}{\int_0^\infty (1 - F_i(t))dt} \quad [44]. \tag{2.13}$$

Garetto *et al.* consider a 2-stage hyper-exponential to determine inter-request time arrivals. They assume the intensities of renewal processes being modulated by a Zipf distribution. Then, they define the rates of exponential stages as $\lambda_{i,1} = \lambda_i z$ and $\lambda_{i,1} = \lambda_i z^{-1}$, in which parameter z applies a temporal locality to this renewal process. The CDF of inter-request times of data item i is then calculated as

$$F_i(t) = 1 - pe^{-\lambda_{i,1}t} - (1-p)e^{-\lambda_{i,2}t}, \tag{2.14}$$

in which $p = z/(z+1)$.

They extend Che's approximation to the renewal traffic model for LRU. They argue that under a general request process $P_i^{in}(\tau)$ and $P_i^{hit}(\tau)$ are not equal since PASTA is not applied anymore. Consequently, to compute $P_i^{in}(\tau)$, they consider that data item i is in the cache at time t if and only if the last request arrived in $[t-\tau, t)$. As a result,

$$P_i^{in}(\tau) = \widehat{F}_i(\tau), \tag{2.15}$$

in which $\widehat{F}_i(t) = \lambda_i \int_0^{\tau} (1 - F_i(t))dt$. On the other hand, when computing $P_i^{hit}(\tau)$, Garetto *et al.* condition on the fact that a request arrives at time t. Thus, the probability that the previous request arrived in $[t-\tau, t)$ is equal to the probability that the last inter-request time is not larger than τ. Therefore,

$$P_i^{hit}(\tau) = F_i(\tau). \tag{2.16}$$

In addition to temporal locality, data items may have different popularity in different geographic locations. Geographical locality of interest means that large-scale systems are typically constructed of smaller heterogeneous communities of users having different interests, and therefore the probability of a request for a given content can vary significantly from region to region. Several factors result in geographic locality of interest. For instance, topics like sports, politics, and news, tend to have localized interest [7, 22]. Another factor is geographic closeness between users, since users close to each other tend to exhibit similarities in language and culture [8, 75, 83]. Geographic locality of interest has also an impact on caching performance [37, 90].

Studies on the distribution of requests in geographically local regions (e.g. number of views of YouTube videos on a university campus) show that the frequency of local requests follow Zipf distributions [47, 61, 100]. In addition, some studies observe that the global frequency of data items (e.g. total number of views of YouTube videos) fit Zipf distributions [23, 82]. Zink *et al.* also observe weak correlation between global and local frequency of data items [100]. Note that studying the users' request patterns at the edge of the network is possible through collecting the users' requests at ISPs' gateways. In addition, the total number of requests for data items (e.g. number of views of YouTube videos) could be collected from the information provided by content publishers (e.g. YouTube servers). On the other hand, there is no study investigating the users' request patterns at the intermediate nodes in the Internet (e.g. combination of users' request patterns of multiple ISP). Therefore, it is implied from these studies that generating synthetic traffic closer to realistic environments should consider Zipf distributions for both local and global frequency of data items as well as weak correlation between them.

While some researchers ignore the geographical locality of request patterns and therefore use synthetic data traffic with identical Zipf request distributions for all regions (e.g. [15, 28, 66, 77, 96]) for modelling

and evaluation of ICN caching, some other researchers apply simple geographical locality in synthetic data traffic. To make different request patterns in different regions, Rossini *et al.* use the following technique [78]: for each new request, they first randomly pick up a data item i according to a global Zipf popularity distribution. Then, they map this new request for this data item to a random user u in the network, attached to an ICN node n. For the next time that data item i is picked up randomly, they select users closer to n as users requesting data item i. The closer user v is to ICN node n, the more probability is to map the request for data item i to v.

Fayazbakhsh *et al.* used a technique that explores the effect of spatial skew [37]. A spatial skew of 0 considers the same global popularity distribution for all regions. On the other hand, the most popular data item in one region may be among the least popular data items in other regions for a spatial skew of 1. Traverso *et al.* use the following technique to apply geographical locality [90]: for the request rate of data item i in region r, they multiply i's overall request rate by $p_{r,i}$, in which $p_{r,i}$ represents the fraction of exogenous requests for i arriving in region r. Traverso's, Fayazbakhsh's and Rossini's approaches to generate traffic with geographical locality do not guarantee that both the global and local distributions have Zipf properties.

2.5 Modelling a Network of Caches

The network of caches constructed in ICNs have attracted renewed interest. By storing data items close to the users, the cost of its retrieval is reduced (ISP's point of view) and the user's quality of experience is improved (user's point of view). The limited caching capacity leads us to two fundamental questions: which data item to evict (replacement mechanism)? and which data item to cache (replication mechanism)? For the first answer, the ICN nodes can use various replacement algorithms such LRU, LFU, k-LRU, etc. In regards to the second question, in-network caching in ICNs are classified into on-path caching and off-path caching based on whether a data item is cached along its delivery path.

In on-path caching, the path is the shortest route from an edge ICN node to the source of data items. Upon the arrival of a data item at intermediate ICN node u, u may cache the data item. ICN node u can make independent decisions on caching the arrived data items. In Leave Copy Everywhere (LCE) replication mechanism for example, ICN node u always stores arrived data items. LCE is easy to implement, but results in lots of redundant copies of a data item in ICN nodes on the delivery path. On the other hand, ICN nodes may collaborate with the other nodes on the delivery path to make replication decision to optimize the placement of a data item on the delivery path. Leave Copy Down (LCD) [56], Move Copy Down (MCD) [57], ProbCache [71] and the age-based caching algorithm proposed by Ming *el al.* [64] are examples of this category.

In LCD, assuming a request for item i hits at ICN node n that is l hops away from the user, a copy of data item i is only cached at node m that is $l-1$ hops from the user. More requests for i in LCD, copies

i at ICN nodes closer to the users. MCD is similar to LCD except that the data item i is evicted from the cache of ICN node in which the hit occurs as it gets cached in ICN node that is $l-1$ hops away from the user. In ProbCache, an ICN node calculates a probability in order to cache a data item. After receiving a data item, ICN node n calculates probability $probCache(x) \approx \frac{x}{c}$ for this data item. Parameter x corresponds to the number of hops (distance) between node n and the content provider and c corresponds to the number of hops between node n and the destination of the data item that is the requesting user for the data item. Parameters x and c are provided in the header of data packets and Interest messages respectively. Parameter c is updated on the path from the requesting user to the content provider. Each ICN node on this path that receives the Interest message, increments parameter c in the header of the Interest message by one. Finally Interest message gets to a content provider. Then, the content provider makes a new copy of requested data item and copies c to its header. The content provider also sets $x = 0$ in the header of the copy of the requested data item. Parameter x on the other hand, is updated on the path from the source node to the requesting user. Each ICN node on this path that receives the data item, increments parameter x in the header of the data packet by one. As a result of $probCache(x)$, data items have more probability to get cached closer to the requesting users. An age-based algorithm proposed by Ming *et al.* assigns a life-time to a data item on its arrival [64]. The data items stays in the cache as long as its life time is not zero. Ming *et al.* allocate longer life-time to more popular data items as well as data items closer to the edge of the network.

In off-path caching [37, 59], ICN nodes cooperate either locally or globally to determine the best content placement, optimized for either latency, network load, or cache utilization. In case of using off-path caching, ICNs use two content discovery/delivery mechanisms either simultaneously or sequentially. One is the standard mechanism explained in Chapter 1. The second of is routing the Interest messages to the ICN nodes that store copies of data items based on off-path caching mechanism. In global off-path caching, data items are replicated according to predefined rules that map data items to caches. Distribute Hash Table (DHT) based routing and caching [53, 81, 99] has been one of the prominent solutions proposed in this domain. Multicache [53] for example, deploys Pastry DHT substrate [80]. In Pastry, each node is assigned an identifier that is chosen uniformly from a flat identifier space. Each node in Pastry has a routing table that enables message forwarding using prefix based routing. At each routing step, a node forwards the message to another node whose identifier shares at least one more digit with the target identifier. To discover a copy of data item i in Pastry, hash of i's name is used to forward the messages for i to a node that possibly has a copy of i cached. In other words, data item i is possibly stored on a node whose identifier is close to the hash value of i's name. In local off-path caching, neighbouring ICNs cooperate to cache data items together. As a result, an ICN node u needs to ask its neighbouring nodes whether they have a copy of data item of interest. Local off-path caching requires significant amounts of overhead. Nearest Replica Routing (NRR) for example, floods messages from ICN node u to the neighbouring ICN nodes in order to discover a data item [37]. The messages have Time To Live (TTL). When an ICN node receives a discovery message, it checks its cache. If the data item is in the cache, the ICN node sends a copy of it back to u. Otherwise, it decreases

the TTL of the message by one and floods the message to its neighbouring nodes if TTL is larger than zero.

Some studies in the past few years have been made to investigate the efficiency of caching in ICN, both experimentally [37, 91] and analytically [66, 71], from which some have inconsistent results. For example, Danzig *et al.* [32] and Rossini *et al.* [79] believe that in-network caching can be more effective. Fayazbakhsh *et al.* [37] and Psaras *et al.* [72] on the other hand believe caching closer to the network edge is more effective. Furthermore, a study by Chai *et al.* shows that selecting only some of the ICN nodes on the delivery path is more profitable [18]. They propose a *centrality-based* caching algorithm that is based on *betweenness centrality* in order to increase the caching performance. Betweenness centrality measures the number of times that an ICN node locates on the content delivery paths between all pairs of ICN nodes in a network topology. There are various definitions of centrality in the literature (e.g. degree, closeness, eigenvector centrality etc.). In the context of on-path caching, the basic intuition is that if an ICN node lies along a high number of content delivery paths (i.e. having high betweenness centrality), then the ICN node's cache is more likely to get a cache hit. By caching only at those more important ICN nodes along the delivery paths, the cache replacement ratio is reduced while still content is cached at ICN nodes where a cache hit is most likely to happen [94].

An analytical investigation of network of caches needs to model the arrival rate of users' requests at the intermediate ICN nodes. Psaras *et al.* propose a Markovian approach to approximate the hit probability in LRU caches under IRM [72]. Their proposed method is based on Markovian assumptions making it difficult to be extended to non-IRM traffic and other cache replacement algorithms. The models proposed by Rosensweig *et al.* [76], Carofiglio *et al.* [15] and Dabirmoghaddam *et al.* [28] rely on the independence assumption among caches, assuming that requests arriving at each cache satisfy the IRM assumptions.

Rosensweig *et al.* for example, propose a-NET, that approximates the miss rates of data items in a network of LRU caches where LCE is used as the cache replication mechanism [76]. Assuming $\lambda^{e}_{v,i}$ and $P_{v,i}$ as the exogenous request rate and hit probability respectively for item i at ICN node v, $\lambda'_{v,i}$ as the miss rate for data item i at node v, their a-NET algorithm finds the request rate for item i at ICN node v, $\lambda_{v,i}$, through equations

$$\lambda_{v,i} = \lambda^{e}_{v,i} + \sum_{u \in R(v)} \lambda'_{u,i} \quad \text{and,} \tag{2.17}$$

$$\lambda'_{v,i} = \lambda_{v,i}(1 - P_{v,i}), \tag{2.18}$$

in which, $R(v)$ is the set of all v's neighbouring ICN nodes from which v may receive a request for i.

Expanding Che's approximation for a network of LRU caches in which LCD is used as the replication mechanism is complicated. Assuming ICN node u as the child of ICN node v, τ_u and τ_v as the characteristic time approximation for ICN nodes u and v respectively, a request for item i at node u at time t is a hit under

IRM, if and only if the previous request arrived in time interval $[t-\tau_u, t]$ and it either was in u's cache or was not in u's cache but was available in v's cache. Consequently, this probability is calculated as

$$P^{hit}_{u,i}(\tau_u, \tau_v) = (1 - e^{-\lambda_{u,i}\tau_u})\big(P^{in}_{u,i}(\tau_u, \tau_v) + (1 - P^{in}_{u,i}(\tau_u, \tau_v))P^{hit}_{v,i}(\tau_u, \tau_v)\big). \tag{2.19}$$

The probability of a hit for data item i at time t at ICN node v under IRM is then

$$P^{hit}_{v,i}(\tau_u, \tau_v) = (1 - e^{-\lambda_{u,i}\left(1-P^{hit}_{u,i}(\tau_u,\tau_v)\right)\tau_v}). \tag{2.20}$$

Under IRM, $P^{hit}_{u,i} = P^{in}_{u,i}$ and $P^{hit}_{v,i} = P^{in}_{v,i}$. A fixed-point iterative procedure is needed to jointly determine $P_{u,i}$ and $P_{v,i}$ since they are interdependent.

During their study, Rosensweig *et al.* identified main potential sources of prediction error that appears between the simulation results and their model: the violation of the IRM (or Poisson) assumption on the miss streams of LRU caches at intermediate ICN nodes [76]. So, (2.17), (2.19) and (2.20) will be an inaccurate approximation of miss rates for i at intermediate ICN node v.

Some studies have shown that TTL-based models for network of caches calculate an accurate approximation of caching behaviour such as hit ratio and miss rate [9, 38, 39, 46, 52]. In a TTL cache, when data item i gets stored in the cache, the cache sets a timer with initial value TTL_i. When this timer expires data item i is evicted from the cache. The timer for data item i is reset to TTL_i if it is requested before its timer expires. Using the Che's approximation [19], the hit provability for data item i would be expressed as

$$P_i(TTL_i) = 1 - e^{-\lambda_i TTL_i}.$$

Garetto's model [44] for a network of caches also has the inaccuracy of Rosensweig's model [76] since the miss streams of requests and aggregated request processes are approximated by Poisson processes. They however, propose a more accurate model to calculate the hit probability of data item i for a network of LRU caches combined with LCE cache replacement algorithm. Assuming ICN node u is the child of ICN node v, they show the arrival process of requests for item i as node v is an ON-OFF Poisson process. In ON phase, i is not cached in u and consequently the requests for i are forwarded to v. In OFF phase however, data item i is cached in u; so no requests for i are forwarded to v.

Assuming τ_u and τ_v as the characteristic time approximation for u and v respectively, Garetto *et al.* observe that a request for i arrives at v at time t only if i is not cached in u before t. This means that no request for i has been arrived at u in interval $[t-\tau_u, t]$. Assuming $\tau_v > \tau_u$, a request for data item i will be hit in v if and only if i has been requested in time interval $[t-\tau_v, t-\tau_u]$. During this time interval, the arrival process for i in not Poisson since it depends on unknown state of ICN node u. Garetto *et al.* however,

assume a Poisson process for the request arrivals of i in this time interval and calculate the hit probability of i at ICN node v as following

$$P_{v,i}^{hit}(\tau_u, \tau_v) \approx 1 - e^{-\lambda'_{u,i}(\tau_v - \tau_u)} \text{ [44]}. \tag{2.21}$$

They also mention that this reasoning cannot be used to calculate $P_{v,i}^{in}$ since data item i may be in v regardless of if it is or is not cached in u. In this regard, they assume

$$P_{v,i}^{in}(\tau_u, \tau_v) \approx 1 - e^{-\lambda'_{u,i}(\tau_v)} \text{ [44]}, \tag{2.22}$$

in order to find τ_v from (2.4). Applying this approximation for a network of non-LRU caches combined with other cache replication mechanism is more complicated. As a result, these TTL-based models for network of caches are computationally costly because of the sophisticated mathematical approach deployed in approximating the cache behaviour.

2.6 Summary

In this chapter, the works proposed for modelling individual caching algorithms under IRM were reviewed; among the existing models, Che's approximation has been found accurate [42, 44] and applicable to other cache replacement algorithms (e.g. FIFO [42], LRU-2Q [48], q-LRU and k-LRU [44]) as well as network of caches [9, 15, 33, 38, 39, 46, 52, 76]. Modelling individual caching algorithms under non-IRM environment, in which temporal and geographical locality is considered in data items' popularity, were also reviewed; while temporal locality has been investigated well in the literature [45, 90, 89], simplifying assumptions are considered to apply geographical locality. Finally, some models proposed for in-network caching in the context of ICN were covered. Rosensweig's model [76] for network of caches is simple but inaccurate because of violation of the IRM assumption between ICN nodes. On the other hand, TTL-based models [9, 38, 39, 44, 52] offer more accurate approximation of caching behaviour in the network of caches, although they are computationally costly.

Chapter 3

Experimental Methodology

This chapter describes the experimental environment used in this thesis to evaluate the performance of caching in ICNs. Section 3.1 introduces ccnSim as a simulation package to simulate ICNs. Section 3.2 briefly overviews features added to ccnSim in order to evaluate the models and algorithms proposed in this thesis. ICN topologies that are studied in this work are covered in Section 3.3. Section 3.4 introduces the parameters used in the models and algorithms of this thesis. Finally, Section 3.5 introduces metrics used here to study the caching performance in ICNs.

3.1 Simulation Tool

To evaluate the proposed models and algorithms, ccnSim [21] is used; ccnSim simulates CCN [49]. This simulation package runs on OMNeT++ which is a C++ object-oriented modular discrete event network simulation framework [93]. OMNeT++ can be used in various network domains such as modelling of wired and wireless communication networks, modelling of queueing networks and modelling of distributed networks.

Simulation environment in OMNeT++ is declared using modules that communicate with message passing. The programmable modules, called simple modules, are written in C++, using the simulation class library. Multiple simple modules could be grouped into compound modules. The main components of OMNeT++ are the following:

1. C++ classes (OMNeT++ class library): extended to customize the simulation environment. One C++ class is required for each new defined simple module. The C++ code implements the behaviour of the simple module.

2. Network Description Language (NED): used to specify the features of new simple and compound modules. In addition, NED is used to declare the interactions between modules.

3. MSG language: used to define format of messages exchanged between network nodes.

The basic features of CCN [21], such as forwarding and caching strategies, cache decision policies and content request model are simulated in ccnSim. The latest released version of ccnSim (ccnSim-$v0.4$) provides two different kinds of content request models:

- IRM: It is the traditional and most used approach in simulation studies. Independent and identically distributed (i.i.d) content requests are generated for a catalog of N contents with fixed popularities following a Poisson process of mean rate equal to λ_i for data item i.

- SNM: It reproduces the model proposed by Traverso *et al.* [90]. SNM takes into account temporal locality of requests through an ON-OFF Poisson process that models the request patterns of different classes of contents.

The directory structure of ccnSim is depicted in Figure 3.1. The *Network* subdirectory contains the NED files describing the topologies (e.g. geant_network.ned, level3_network.ned etc.). The *Module* subdirectory contains NED files describing the simple and compound modules; for example *node.ned* describes a CCN node in ccnSim as shown in Code C.1. Compound *node.ned* has three submodules (lines 20 to 22): *core_layer* that implements the content discovery/delivery process; *strategy_layer* that implements content advertisement which is responsible for filling up the FIBs for ICN nodes; and *content_store* that implements cache replacement and replication algorithms. The NED files also declare the connection between the submodules (lines 29 and 30 in Code C.1) as well as the features of the module (lines 10 to 13 in Code C.1). The *Packet* subdirectory contains *ccn_data.msg* and *ccn_interest.msg* as the two message types in CCN. The C++ code for simples modules exist in the *src* directory. This directory itself contains the following subdirectories (right side of Figure 3.1): *clients* in which different kinds of clients such as *client_IRM.cc* and *client_ShotNoise.cc* are implemented; *content*: this module is responsible for defining the distribution based on which the clients generate their requests; and *node* that implements the core layer, strategies and cache replacement and replication algorithms.

3.2 Implementation

Modifications were made to the code of ccnSim (illustrated in italic and red in Figure 3.1). The 2-LRU and LRU-2 algorithms have been implemented since these two algorithm are studied in details in Chapter 4. The proposed traffic generator in Chapter 5 is implemented through additional simple modules implemented in *client_GeographicalLocality.cc* and *GeographicalLocalityContentDistribution.cc* files. A new core_layer module is also introduced that apply the local search in the process of content discovery/delivery (explained in Chapter 5); this module is implemented in *core_ls_layer.cc*. This new module is responsible to handle the local search messages, PLSTs, timers and local search replies, as explained in Section 5.3.

The settings for the experimental environment in OMNeT++ are configured in a file named *omnetpp.ned* (illustrated in Appendix C). The settings can be divided into the following categories:

- Topology settings: as shown in Code C.2, this category configures settings related to the topology of CCN system, such as the network (e.g. Dtelecom, Geant, etc.), list of repositories and the number of repositories.

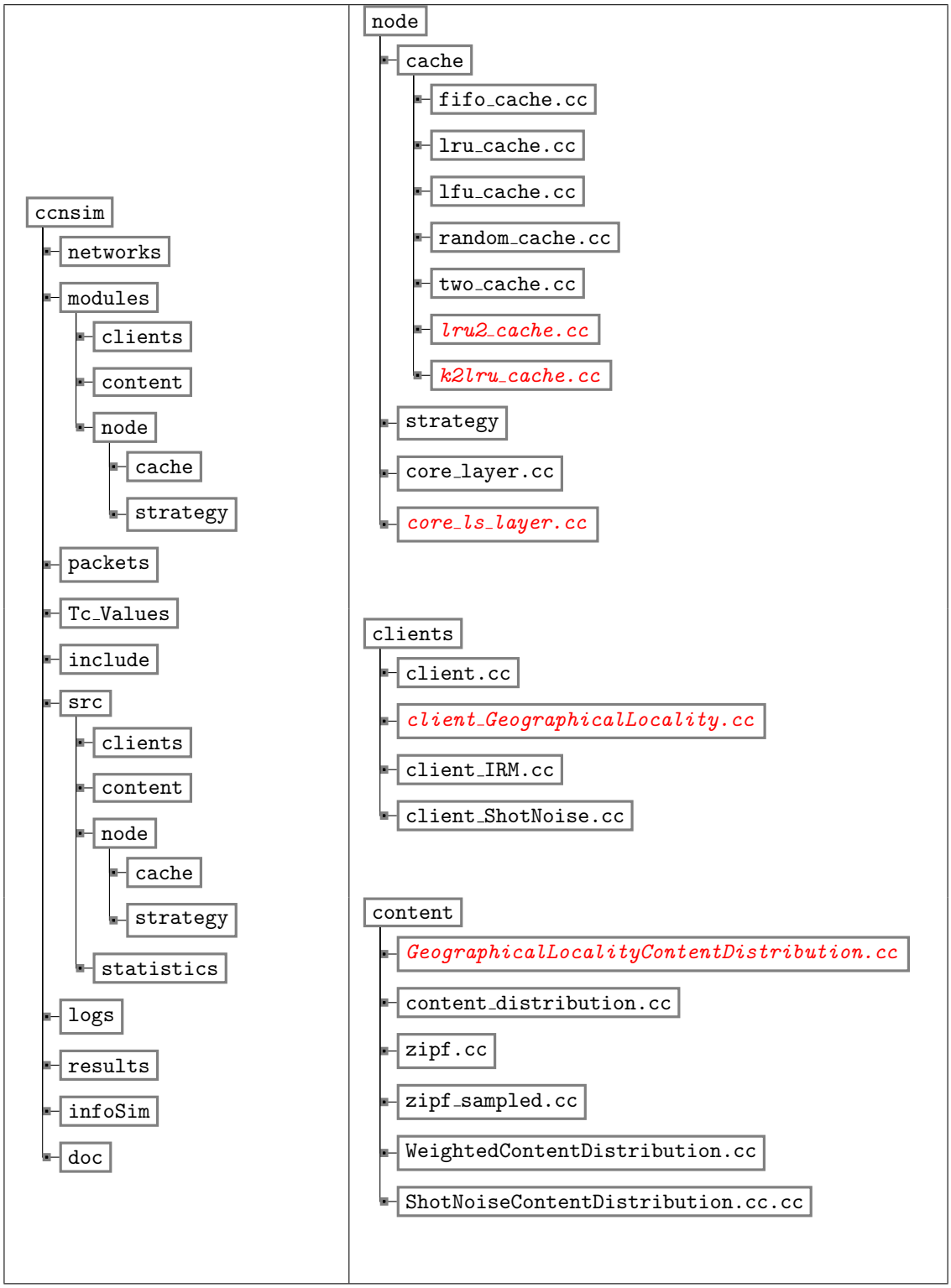


Figure 3.1: ccnSim directory structure.

- Client settings: as shown in Code C.3, the category configures settings related to the clients such as list of clients, number of clients, the arrival rate and client type. A user's requests for a data item is assumed to arrive at an ICN node as a Poisson process.

- Local search settings: this category configures the settings of the proposed traffic generator in Chapter 5 (Code C.4).

- Content distribution settings: this category configures the settings related to the content distribution such as: the file size, Zipf parameter, number of data items in the simulation, type of content distribution, and the parameters related to the proposed traffic generator in Chapter 5 (Code C.5).

- Forwarding settings: this category sets the algorithm used for Interest forwarding (Code C.6), such as Shortest Path Routing (SPR) and Nearest Replica Routing (NRR) [37].

- Caching settings: this category sets the parameters related to the caching such as cache replacement algorithms (e.g. LRU, LFU, FIFO, LRU-2, 2-LRU, q-LRU and random), cache replication algorithms (e.g. LCE and LCD) and the cache size of the CCN nodes (Code C.7).

- Simulation run-time: ccnSim collects the statistics (e.g. average distance, hit ratio etc) after the simulation environment enters into a steady state. First, ccnSim waits for all the caches in the network to get full. Then, ccnSim waits for the simulation environment to get stable. The Stabilization happens when the variance of hit probability at each node in the network goes below a threshold. At this point, ccnSim starts collecting the statistics. After the system gets stable, the simulation runs to completion. The simulation run-time is selected such that the least popular items are requested at least once. For example, assume the popularity of 20000 data items follow a Zipf distribution with $\alpha = 1.4$ and the overall request arrival rate at an ICN cache is modelled as a Poisson process with $\lambda = 2$ requests/second. Therefore, the least popular data item is requested once every 1.6e+06 seconds on average. The simulation run-time is then set to 9e+06 (representing 104 days) in order to request the least popular item one or more times.

3.3 Network Topologies

The topologies used to evaluate the performance of caching in ICNs are either k-array trees [18, 28, 72, 71] or ISP topologies [10, 11, 73, 77, 86]. To have a more realistic study of the behaviour of in-network caching in ICNs, four publicly available ISP topologies (available in ccnSim), shown in Table 3.1 are used in this thesis, as used in other studies [10, 11, 77, 86]. The ccnSim simulator is configured such that one of the ICN nodes in each topology is selected as the gateway node of that topology. The out-going traffic at this gateway node contributes to the load on the source node that hosts all the data items. For a given topology, ccnSim then creates a tree from the gateway node to other nodes in the topology.

Table 3.1: Specification of topologies.

	specifications				
name	*inter-nodes*	*edge-nodes*	*depth*	*max degree*	*average degree*
Level3	5	41	5	29	9.00
Dtelecom	7	61	4	52	9.57
Tiger	12	10	5	4	1.75
Geant	12	10	6	4	1.75

3.4 Configuration Parameters

Table 3.2 depicts the notations used in the thesis. First, the pattern of data items arriving at a cache under IRM is described here. IRM assumes that requests are generated for a fixed catalogue size of data items, shown by N. IRM also assumes that request rate for data item i arriving at node k is constant over time and follows a Poisson process with rate $\lambda_{k,i}$. The request probability of data items at ICN node k are assumed to follows a Zipf distribution with parameter α_k. This results in $p_{k,i} \approx (1/\Pi_{k,i})^{\alpha_k}$, in which $\Pi_{k,i}$ indicates the rank of data item i's popularity at ICN node k. Then, assuming λ_k as the total request rate of all data items at node k, $\lambda_{k,i}$ is calculated as $\lambda_{k,i} = p_{k,i}\lambda_k$. Under non-IRM however, $\lambda_{k,i}$ is calculated using (2.13). Parameter z indicates the strength of temporal locality in a 2-stage hyper-exponential distribution in (2.14).

Assuming $P_{k,i}$ as the hit probability of a request for data item i at ICN node k, the miss rate of data item i at ICN node k is calculated as $\lambda'_{k,i} = (1 - P_{k,i})\lambda_{k,i}$. Consequently, the overall miss rate and hit probability at ICN node k are calculated as $\lambda'_k = \sum_{i=1}^{N} \lambda'_{k,i}$ and $P_k = 1 - \lambda'_k/\lambda_k$, respectively. C_k is the cache size of ICN node k which is used in (2.4) to calculate τ_k; as a result τ_k is a function of C_k that is indicated by $\tau_k(C_k)$. Chapter 4 assumes identical distributions for all ICN nodes meaning the followings:

$$\Pi_{k,i} = i, \forall k \in R, \text{ and}$$
$$p_{k,i} = p_{l,i}, \forall k, l \in R, l \neq k.$$

In this regard, Chapter 4 uses notations p_i, λ_i, λ'_i for the sake of simplicity. Other notations in Table 3.2 are described in the chapters in which they are used.

3.5 Metrics

To study the caching performance in ICNs, the following metrics are considered in this thesis:

- **Hit ratio/miss rate**: a good caching mechanism in an ICN network maximizes the overall hit ratio in the network. The overall miss rate in an ICN network is a fraction of requests that are not fulfilled inside the ICN network. The miss rate of an ICN network is proportional with the hit ratio in that ICN network.

Table 3.2: Notations

	General	
N	$\triangleq$	number of data items
s	$\triangleq$	source of data items
λ_k	$\triangleq$	arrival rate at ICN node k
$p_{k,i}$	$\triangleq$	item i's popularity at ICN node k
$\lambda_{k,i}$	$\triangleq$	arrival rate of data item i at ICN node k
$P_{k,i}$	$\triangleq$	hit probability of data item i at ICN node k
P_k	$\triangleq$	the overall hit probability of data items at ICN node k
$\lambda'_{k,i}$	$\triangleq$	miss rate of data item i at ICN node k
λ'_k	$\triangleq$	the overall miss rate of data items at ICN node k
α_k	$\triangleq$	Zipf parameter of the traffic at ICN node k
$\tau_k(C_k)$	$\triangleq$	characteristic time approximation of the cache at ICN node k, that is a function of C_k
	Chapter 5	
t	$\triangleq$	a threshold for Zipf match
ψ	$\triangleq$	a parameter to determine the similarity between two distributions
S_k^λ	$\triangleq$	set of $\lambda_{k,i}$, $1 \leq i \leq N$: $\{\lambda_{k,1}, \lambda_{k,2} \dots \lambda_{k,N}\}$
$\Pi_{k,i}$	$\triangleq$	rank of item i's popularity in region k
$\Lambda_{k,j}$	$\triangleq$	request rate of an item with rank j in region k
Π_k	$\triangleq$	function mapping item i to $\Pi_{k,i}$ $1 \leq i \leq N_k$
Γ_k	$\triangleq$	set of data items in region k
	Chapter 6	
z	$\triangleq$	temporal locality parameter
D_k	$\triangleq$	the shortest distance between k to s
k^i	$\triangleq$	i'th immediate parent of k, e.g. k^0 and k^1 are node k itself and i's parent respectively
	Common in Chapters 5 and 6	
R	$\triangleq$	set of all ICN nodes
E	$\triangleq$	set of all ICN edge nodes
C	$\triangleq$	the overall cache budge; $C = \sum_{\forall r \in R} C_k$
D	$\triangleq$	the depth of a tree
S_k	$\triangleq$	set of all ICN nodes in a subtree rooted at k
N_k	$\triangleq$	number of arriving data items at ICN node k
C_k	$\triangleq$	size of cache allocated to ICN node k
R_k	$\triangleq$	set of all children of ICN node k

- **Load on the server**: the miss request leaving the gateway of a topology in an ICN network is considered as the load on the server that hosts all data items. Lower miss rate at the root results in lower load on the server.

- **The average retrieval distance**: this metric defines the average distance (number of hops) between the users and the first copy of data item of their interest. This metric is considered as the latency that users feel in order to access the data items. It is assumed in this thesis that the all hops in an ICN network have the same latency. Caching in ICNs helps minimizing this distance (latency).

Chapter 4

Modelling LRU-2

4.1 Motivation

ICNs are comprised of nodes that are equipped with cache storages. These connected ICN nodes then construct a hierarchy of caches. Studying the performance of caching in a large hierarchy of caches through simulations is extremely costly. Garetto *et al.* for example found that investigating the caching performance in an ICN with 1365 nodes through simulations is very expensive since the simulation needs lots of memory, high CPU usage and long time to enter into the steady state [44]. Modelling of such a large cache network however needs the same computation cost of small cache networks.

Modelling the caching performance in a network of caches in ICNs requires an accurate approximation for the performance of each individual cache in the network. Calculating an accurate approximation of LRU performance has been interesting for researchers [19, 30, 72] since LRU is used in shaping other cache replacement algorithms with higher hit ratios such as q-LRU, LRU-k [68], ARC [63], LRU-2Q [51] and k-LRU [44]. LRU-k is the oldest one in the family of LRU algorithms that outperforms LRU through considering recency and frequency together in its eviction mechanism. LRU-2Q then was proposed to overcome the run-time complexity of LRU-k [51]. The k-LRU cache replacement algorithm is the latest one in the family of LRU caching algorithms that outperforms LRU through deploying an eviction mechanism with the same idea used in LRU-k, considering both frequency and recency, with a low run-time complexity. Besides the studies that modelled different cache replacement algorithms such as FIFO [30, 44], LRU [19, 30, 72], q-LRU [44], LRU-2Q [48] and k-LRU [46, 44], there is no study that modelled LRU-k. The main contribution in this chapter is modelling LRU-2, using Che's approximation, explained in Section 2.4, as a specific case of LRU-k for $k = 2$. The experiments show that the proposed model approximates the LRU-2 algorithm accurately. The closest work to this modelling is Boyar's work that compares the performance of LRU-k versus LRU [14]. They however, do not find the miss rate for LRU-k caches.

The rest of this chapter is organized as follows. First, LRU-2 is mathematically modelled for an individual cache and hierarchical trees of caches in Section 4.2. The model is validated analytically and with simulation in Section 4.3. First, the LRU-2 model is evaluated in a simple binary tree with three nodes. The accuracy of the model is studied for individual nodes at the edge and the root of such topology. Then, the model is investigated in four publicly available ISP topologies. This section also compares LRU-2 with 2-LRU [44].

Section 4.4 summarizes the chapter.

The author would like to thank Nicholas Beaton from the Department of Math and Stats that collaborated in solving the derivation and verification of the LRU-2 model. Note that portions of this chapter have been accepted for publication at *LCN 2018*.

4.2 The LRU-2 Model

O'Neil *et al.* proposed LRU-k in order to take into account the history of the last k references to each data item [68]. In this regard, the LRU-k cache stores k timestamps for each data item. A buffer is used to store the timestamps of not only cached data items but also the evicted data items. Upon replacement, the data item with the oldest k^{th} timestamp will be evicted. In their LRU-k caching algorithm, however, a request for a data item with a cache miss will be stored regardless of its timestamp.

For data item i, LRU-2 model gets the number of data items (N), the request rate for i (λ_i) and the cache size (C) and calculates the miss rate for data item i (λ_i'). In this regard, Che's approximation is extended to model the miss rate of data items in a LRU-2 cache. To adopt Che's approximation for LRU-2, we need to redefine τ_i as follows: τ_i is the time needed before C distinct data items (not including item i) are requested *twice* by the users since the second most recent request for item i. As a result, data item i is in the cache at time t if and only if less than τ_i has elapsed since the second most recent request for item i.

The arrival process of item i is presented in Figure 4.1. Assuming item i gets cached at t_0, a request for it at time t_l is a cache hit if and only if $t_l - t_{l-2} < \tau_i$. A request for item i is a cache miss at t_n since $t_n - t_{n-2} > \tau_i$. As described before, item i cannot return to the cache if there is no other data item in the cache whose timestamp is older than i's timestamp. Item i gets stored again however at time t'_m since $t'_m - t'_{m-2} < \tau_i$. There is one cache miss in interval $(t_0, t_n]$, while all requests for item i end in cache miss in interval $(t'_0, t'_m]$. We calculate the expected value of item i's miss rate to be

$$\lambda_i' = \frac{1 + (\lambda_i \mathbb{E}[t'_m] - 1)}{\mathbb{E}[t_n] + \mathbb{E}[t'_m]} = \frac{\lambda_i \mathbb{E}[t'_m]}{\mathbb{E}[t_n] + \mathbb{E}[t'_m]}. \tag{4.1}$$

4.2.1 Calculating τ_i

As with LRU, we let $u_l = t_l - t_{l-1}$. For item i, $\{u_1 + u_2, u_2 + u_3, \dots\}$ are independent and identical Erlang distributed random variables with pdf of $f(t, \lambda_i) = \lambda_i^2 t e^{-\lambda_i t}$. Consequently, the time-average probability P_i that data item i is in the cache is then given by

$$P_i(\tau_i) = 1 - e^{-\lambda_i \tau_i}(1 + \lambda_i \tau_i). \tag{4.2}$$

If we again suppose that τ_i is independent of i, we should expect $\tau \equiv \tau_i$ to satisfy

$$\sum_{j=1}^{N} P_j(\tau) = \sum_{j=1}^{N} 1 - e^{-\lambda_j \tau}(1 + \lambda_j \tau) = C. \tag{4.3}$$

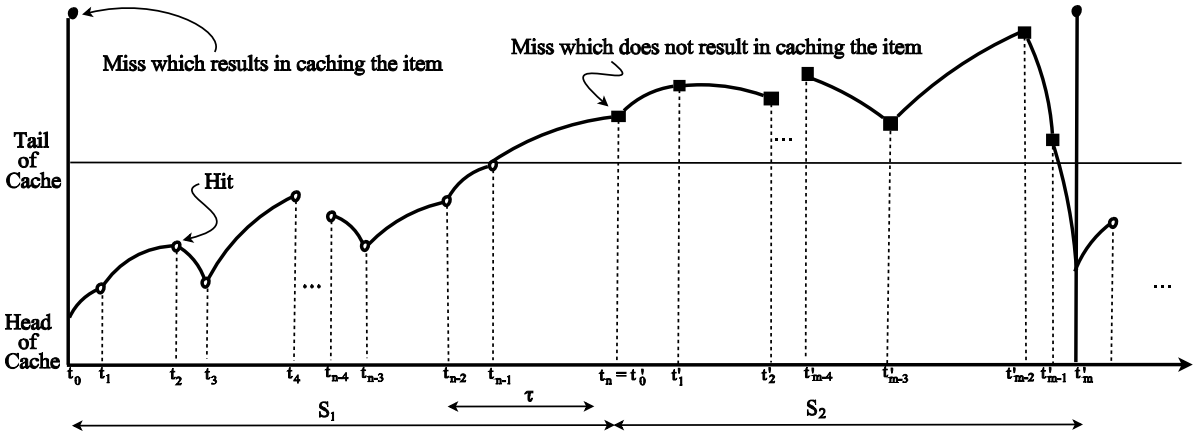


Figure 4.1: Arrival processes of data item i at a LRU-2 cache.

4.2.2 Calculating t_n

Using a similar setup as for LRU, we now let n be the smallest value such that $t_n - t_{n-2} = u_n + u_{n-1} > \tau$, as shown in Figure 4.1. For item i, we now have a sequence of exponential random variables $u_1, u_2, u_3 \ldots$ and we terminate when two consecutive values add to more than τ. Moreover, we will assume that there is a special value x which *precedes* the sequence, and we terminate after u_1 if $u_1 + x > \tau$. Let $S_1(x)$ be the sum, conditioned on the special value x.

Now we sample u_1 with expected value $\frac{1}{\lambda_i}$. If $x > \tau$, we terminate there, regardless of the value of u_1, so this is not interesting. If $x < \tau$, then with probability $e^{-\lambda_i(\tau - x)}$ we also terminate at u_1. Otherwise, we must keep going, but now what happens afterwards depends on u_1. We must integrate over the possible values of u_1, to obtain

$$\mathbb{E}[S_1(x)] = \frac{1}{\lambda_i} + \int_0^{\tau - x} \lambda_i e^{-\lambda_i u} \mathbb{E}[S_1(u)] du. \tag{4.4}$$

Let $f(x) = \mathbb{E}[S_1(x)]$, and differentiate with respect to x to obtain

$$f'(x) = -\lambda_i e^{-\lambda_i(\tau - x)} f(\tau - x). \tag{4.5}$$

Solving (4.5), as shown in Appendix A, results in

$$\mathbb{E}[S_1(x)] = \frac{\left(\frac{1}{\sqrt{\mu_1}} e^{\mu_1 (x - \frac{\tau}{2})} - \frac{1}{\sqrt{\mu_2}} e^{\mu_2 (x - \frac{\tau}{2})} \right)}{\lambda_i \left(\frac{1}{\sqrt{\mu_1}} e^{\mu_1 \frac{\tau}{2}} - \frac{1}{\sqrt{\mu_2}} e^{\mu_2 \frac{\tau}{2}} \right)}, \tag{4.6}$$

in which $\mu_{1,2}$ depend on λ_i and τ and are defined in Appendix B (specifically (A.8)).

Next, assume that item i has a request at time $t = t_0$, at which point it becomes cached, after being evicted. We wish to determine the expected time until it is evicted again. This will depend on the last request before t_0, ie. t_{-1}. Unfortunately, $u_0 = t_0 - t_{-1}$ is not fixed, nor is it completely random; its distribution is

affected by the fact that i is cached at t_0, but evicted at t_{-1}. We first determine the distribution of u_{-1}, initially conditioning only on $u_{-1} + u_{-2} > \tau$. By Bayes' theorem, this distribution satisfies

$$\begin{aligned} f^*_{u_{-1}}(u) &= \frac{\lambda_i e^{-\lambda_i u}\mathbb{E}[u_{-1} + u_{-2} > \tau | u_{-1} = u]}{\int_0^\infty \lambda_i e^{-\lambda_i v}\mathbb{E}[u_{-1} + u_{-2} > \tau | u_{-1} = v]dv} \\ &= \begin{cases} \frac{\lambda_i}{1+\tau\lambda_i} & u < \tau \\ \frac{\lambda_i e^{\lambda_i(\tau-u)}}{1+\tau\lambda_i} & u > \tau. \end{cases} \end{aligned} \tag{4.7}$$

Given the distribution of u_{-1}, we can get the distribution of u_0 in the same way, now taking into account $u_0 + u_{-1} < \tau$:

$$\begin{aligned} f_{u_0}(u) &= \frac{\lambda_i e^{-\lambda_i u}\mathbb{E}[u_0 + u_{-1} < \tau | u_0 = u]}{\int_0^\infty \lambda_i e^{-\lambda_i v}\mathbb{E}[u_0 + u_{-1} < \tau | u_0 = v]dv} \\ &= \frac{\lambda_i^2 e^{-\lambda_i u}(\tau - u)}{\lambda_i\tau + e^{-\lambda_i\tau} - 1} \qquad \text{for } 0 < u < \tau. \end{aligned} \tag{4.8}$$

Hence, $\mathbb{E}[t_n]$ can be obtained through (4.9).

$$\mathbb{E}[t_n] = \int_0^\tau f_{u_0}(u)\mathbb{E}[S_1(u)]du = \frac{e^{\frac{3}{2}\lambda_i\tau}\Big(e^{-\lambda_i\tau}(\mu_1^{\frac{3}{2}}e^{\mu_1\frac{\tau}{2}} - \mu_2^{\frac{3}{2}}e^{\mu_2\frac{\tau}{2}}) + \mu_2^{\frac{3}{2}}e^{-\mu_2\frac{\tau}{2}}(1-\mu_1\tau) - \mu_1^{\frac{3}{2}}e^{-\mu_1\frac{\tau}{2}}(1-\mu_2\tau)\Big)}{\lambda_i^2(\lambda_i\tau + e^{-\lambda_i\tau} - 1)(\sqrt{\mu_2}e^{\mu_1\frac{\tau}{2}} - \sqrt{\mu_1}e^{\mu_2\frac{\tau}{2}})}. \tag{4.9}$$

4.2.3 Calculating t'_m

Now say item i was requested at time $t = t'_0$, and at that point it was not in the cache. It then has requests at times $t'_1, t'_2, t'_3, \ldots$, with m the smallest value such that $t'_m - t'_{m-2} = u'_m + u'_{m-1} < \tau$. We again have a sequence of exponential random variables $u'_1, u'_2, u'_3, \ldots$, but this time we terminate when two consecutive values add to less than τ. As before, we define a special value x which precedes the sequence, allowing us to terminate after u'_1 if $u'_1 + x < \tau$. Let $S_2(x)$ be the sum of the u'_i, conditioned on the special value x. We sample u'_1 with expected value $\frac{1}{\lambda_i}$. If $x > \tau$, we continue and sample u'_2, regardless of u'_1's value (see Appendix B). If $x < \tau$, with probability $1 - e^{-\lambda_i(\tau-x)}$ the process terminates after u'_1. Otherwise we keep going, but now what happens after depends on the value of u'_1, so we integrate over the possible values, to get

$$\mathbb{E}[S_2(x)] = \frac{1}{\lambda_i} + \int_{\tau-x}^\infty \lambda_i e^{-\lambda_i u}\mathbb{E}[S_2(u)]du. \tag{4.10}$$

Let $f(x) = \mathbb{E}[S_2(x)]$, and differentiate with respect to x to get

$$f'(x) = \lambda_i e^{-\lambda_i(\tau-x)} f(\tau - x). \tag{4.11}$$

Solving (4.11), as shown in Appendix B, results in

$$\mathbb{E}[S_2(x)] = \begin{cases} \frac{f(x)}{\lambda_i(1-e^{-\tau\lambda_i}-Q)f(\tau)} & x < \tau \\ \frac{1}{\lambda_i(1-e^{-\tau\lambda_i}-Q)} & x \geq \tau, \end{cases} \tag{4.12}$$

where $f(x)$ and Q are defined in (B.1) and (B.5) respectively.

Next, suppose i is requested at $t = t'_0$ and i has been evicted before t'_0. That is, $u'_0 + u'_{-1} > \tau$ while $u'_{-1} + u'_{-2} < \tau$. We first determine the distribution of u'_{-1}, conditioned only on $u'_{-1} + u'_{-2} < \tau$. By Bayes' theorem, this distribution satisfies

$$\begin{aligned} f^*_{u'_{-1}}(u) &= \frac{\lambda_i e^{-\lambda_i u}\mathbb{E}[u_{-1} + u_{-2} < \tau | u_{-1} = u]}{\int_0^\infty \lambda_i e^{-\lambda_i v}\mathbb{E}[u_{-1} + u_{-2} < \tau | u_{-1} = v]dv} \\ &= \frac{\lambda_i(e^{-\lambda_i u} - e^{-\lambda_i \tau})}{1 - (1 + \lambda_i\tau)e^{-\lambda_i\tau}} \qquad \text{for } 0 < u < \tau. \end{aligned} \tag{4.13}$$

Given the distribution of u'_{-1}, we again use Bayes' theorem to consider $u'_0 + u'_{-1} > \tau$ to get the distribution of u'_0:

$$\begin{aligned} f_{u'_0}(u) &= \frac{\lambda_i e^{-\lambda_i u}\mathbb{E}[u_0 + u_{-1} > \tau | u_0 = u]}{\int_0^\infty \lambda_i e^{-\lambda_i v}\mathbb{E}[u_0 + u_{-1} > \tau | u_0 = v]dv} \\ &= \begin{cases} \frac{\lambda_i e^{-\lambda_i u}(-\lambda_i u + e^{\lambda_i u} - 1)}{\lambda_i\tau + e^{-\lambda_i\tau} - 1} & u < \tau \\ \frac{\lambda_i e^{-\lambda_i u}(-\lambda_i \tau + e^{\lambda_i \tau} - 1)}{\lambda_i\tau + e^{-\lambda_i\tau} - 1} & u > \tau. \end{cases} \end{aligned} \tag{4.14}$$

Hence, $\mathbb{E}(t'_m)$ is calculated through (4.15).

$$\mathbb{E}(t'_m) = \int_0^\tau f_{u'_0}(u)\mathbb{E}[S_2(u)]du = \frac{\frac{e^{\lambda_i\frac{\tau}{2}}\left(\sqrt{u_2}(\mu_1\lambda_i\tau - \mu_2)e^{\mu_1\frac{\tau}{2}} + \sqrt{\mu_1}(\mu_2\lambda_i\tau - \mu_1)e^{\mu_2\frac{\tau}{2}}\right)}{\left(\lambda_i^2 f(\tau)\right)} + e^{-\lambda_i\tau}(-\lambda_i\tau + e^{\lambda_i\tau} - 1)}{\lambda_i(\lambda_i\tau + e^{-\lambda_i\tau} - 1)(1 - e^{-\lambda_i\tau} - Q)}. \tag{4.15}$$

4.3 Model Validation/Insights

The goal of this section is threefold. First, the previously derived analytical expressions are validated against simulations to investigate accuracy of the proposed model based on different system/traffic parameters. Second, LRU-2 model and 2-LRU are studied. Finally, the proposed model is studied in realistic topologies of Section 3.3.

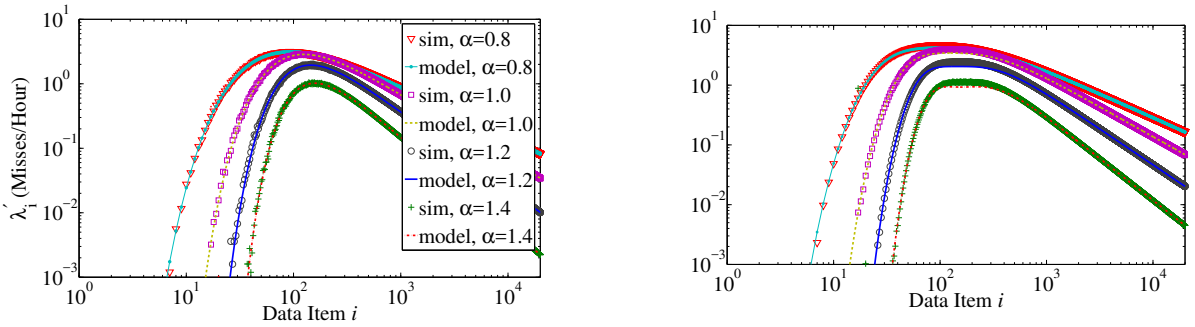
All caches have the same size, and caches connected to direct users have the same exogenous request pattern. The request arrival pattern at cache c (exogenous traffic), is modelled as a Poisson process with parameter $\lambda^e_c = \sum_{\forall i} \lambda^e_{c,i} = 2$. Item i's request probability (p_i), follows a Zipf distribution with parameter α ($p_i \propto \frac{1}{i^\alpha}$). The request rate for data item i will thus be $\lambda^e_{c,i} = \lambda^e_c p_i$. Having $N = 20000$, (4.1) is used to evaluate the proposed model. The following two scenarios are taken into account in the experiments:

- Scenario 1: $C = 200$, $\alpha \in \{0.8, 1.0, 1.2, 1.4\}$,
- Scenario 2: $\alpha = 1.0$, $C \in \{200, 500, 1000, 2000\}$.

The simulation results are the average of five simulation runs for each case study in these two scenarios.

4.3.1 Evaluation of the LRU-2 Model

In this section, a simple topology of a binary tree with three caches is studied. Two edge caches $\{c_1, c_2\}$ are connected to the users while the root cache r gets no direct requests. Here, the performance of caches are studied through their overall miss rates. The overall miss rate at a node contributes to the load on the server as mentioned in Section 3.5. In this regard, the miss rate per data item measured by simulations and estimated by the proposed model are compared.



(a) Edge caches **(b)** Root cache

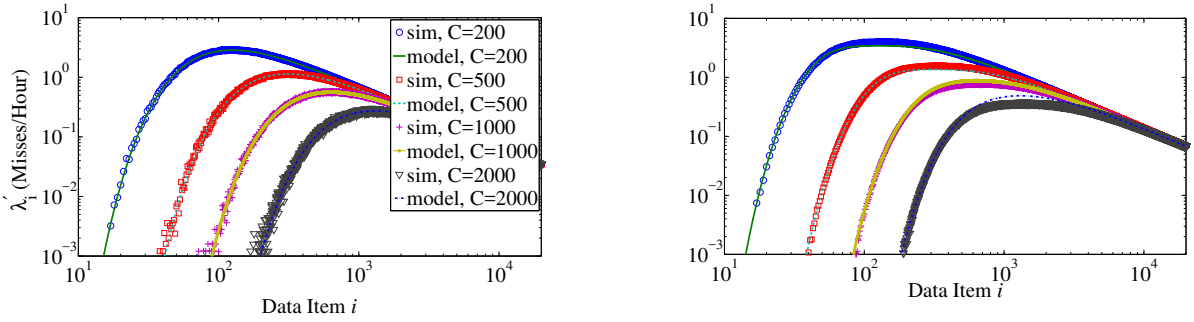
Figure 4.2: LRU-2, $C = 200$ (log-log scale).

For Scenario 1 (see Figure 4.2a), τ_{c_1} and τ_{c_2} are calculated using (4.3) to get the miss rate for each item. For edge caches, $\lambda_{c_1,i} = \lambda_{c_2,i} = 2p_i$ since they have no child caches. This results in $\tau_{c_1} = \tau_{c_2} = 969, 1056, 1554, 2963$ for the four different values of α respectively. For $\alpha = 1.0$ for example, this means that if the requests arrive with rate of 2 requests per second, a request for data item i in LRU-2 cache with capacity of 200 at time t is a hit if the time interval between t and the time at which the second most recent request in the past for i took place is shorter than 1056 seconds.

Then, $t_{c_1,n}$ and $t_{c_2,n}$ via (4.9) and $t'_{c_1,m}$ and $t'_{c_2,m}$ via (4.15) are calculated, in order to find $\lambda'_{c_1,i}$ and $\lambda'_{c_2,i}$ using (4.1). The request rate for item i at the root cache r is calculated through (2.17) which results in $\lambda_{r,i} = \lambda'_{c_1,i} + \lambda'_{c_2,i}$ since there is no exogenous request traffic for the root cache. Then, this value is used to find τ_r. Having τ_r calculated, item i's average miss rate at the root cache can be approximated as represented in Figure 4.2b. The proposed model for LRU-2 provides a good approximation of the cache miss rate per data item. Similar steps are follows for Scenario 2 as well. This scenario is evaluated in Figure 4.3 where the cache miss rate decreases as the cache size increases.

4.3.2 LRU-2 Model vs 2-LRU Evaluation

In this section, the LRU-2 and 2-LRU caching algorithms are investigated with respect to performance in terms of the miss rates at each level in the cache hierarchy. The results of LRU-2 simulations are removed from the study since LRU-2 simulations match the LRU-2 model precisely. In addition, Gast's approximation for 2-LRU in (2.10) [46] is used to approximate the performance of 2-LRU algorithm.



(a) Edge caches **(b)** Root cache

Figure 4.3: LRU-2, $\alpha = 1.0$ (log-log scale).

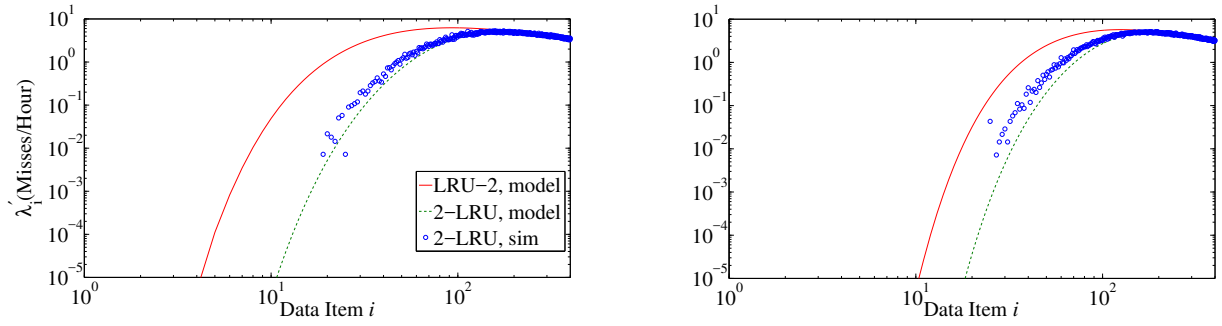
The comparison between 2-LRU and LRU-2 for edge nodes in Scenario 1 is demonstrated in Figure 4.4. The x-axis is cut off at $2C$, since all the remaining data items have 100% miss ratios for both models. The miss rates for popular items is very low as they are almost always in the cache. This figure shows that miss rates in 2-LRU get closer to the miss rates in LRU-2 as α increases. In addition, Gast's model underestimates the miss rate for items in 2-LRU caching algorithm as α increases. The same general behaviour at the root cache is observed as shown in Figure 4.5. The comparison for Scenario 2 is depicted in Figures 4.6 and 4.7. As the cache size increases, Gast's model for 2-LRU slightly underestimates the miss rates.

Thus, it can be concluded that 2-LRU outperforms LRU-2. Moreover, Gast's approximation for 2-LRU in (2.10) underestimates the miss rates in 2-LRU for larger α and cache size. In contrast, the proposed model for LRU-2 calculates a better approximation of 2-LRU behaviour for larger α and cache size.

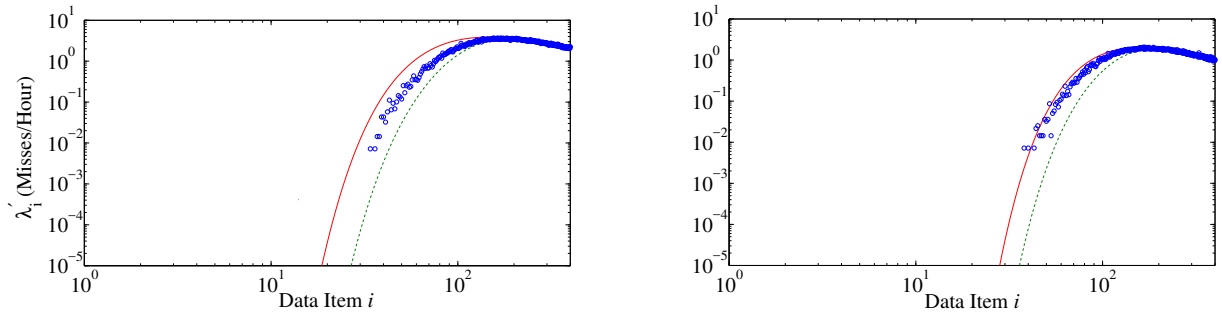
4.3.3 Realistic Topologies

In this section, the caching performance in realistic topologies is investigated. Only edge caches are directly connected to users. The intermediate caches receive endogenous requests from their children caches. The overall hit ratio of the network is calculated as $\left(\sum_{\forall c}\sum_{\forall i}(\lambda_{c,i} - \lambda'_{c,i})\right)/\left(\sum_{\forall c}\sum_{\forall i}\lambda_{c,i}\right)$. Table 4.1 gives hit probability predictions for 2-LRU and LRU-2 in Geant topology for Gast's 2-LRU model and our LRU-2 model respectively, and the corresponding simulations. 2-LRU provides a slightly higher overall hit ratio in both simulation and modelling. Gast's 2-LRU model has an error between $(+0.44\%, +5.03\%)$ for the overall hit ratio, while the proposed LRU-2 model has an error between $(-0.37\%, 6.05\%)$. The results for Tiger (Table 4.2), DTelecom (Table 4.3) and Level3 (Table 4.4) confirm this difference.

For larger cache sizes, the 2-LRU model continues to overestimate and the LRU-2 model is stable in nearly all cases. For smaller values of α, both algorithms are less accurate. The last column in Tables 4.1 to 4.4 shows the fit between the proposed model for LRU-2 and the simulation results of 2-LRU. As in the synthetic cases, the LRU-2 model approximates a better hit ratio for 2-LRU for larger cache and large α. For Dtelecom topology for example (Table 4.3), the proposed model for LRU-2 approximates the hit ratio

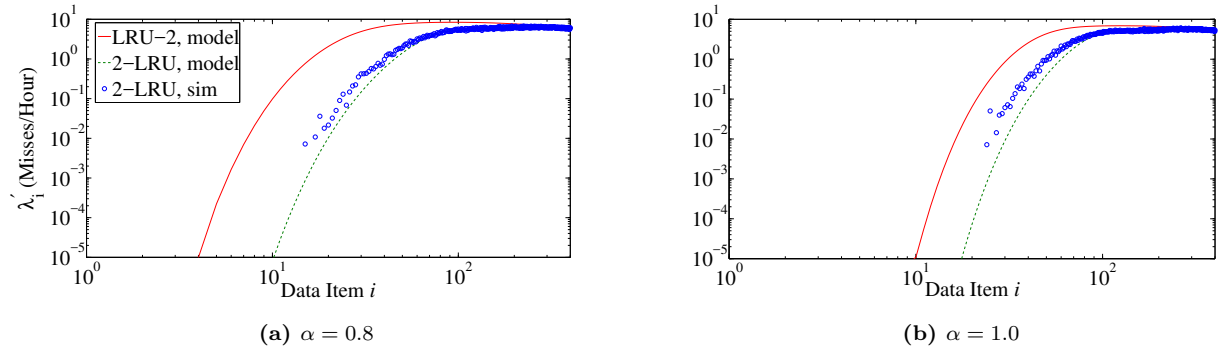


(a) $\alpha = 0.8$ **(b)** $\alpha = 1.0$

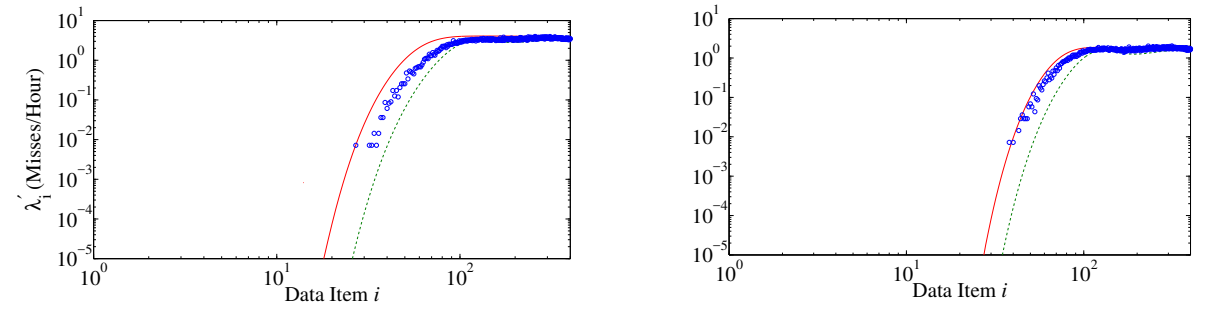


(c) $\alpha = 1.2$ **(d)** $\alpha = 1.4$

Figure 4.4: LRU-2 *vs* 2-LRU, $C = 200$, edge.

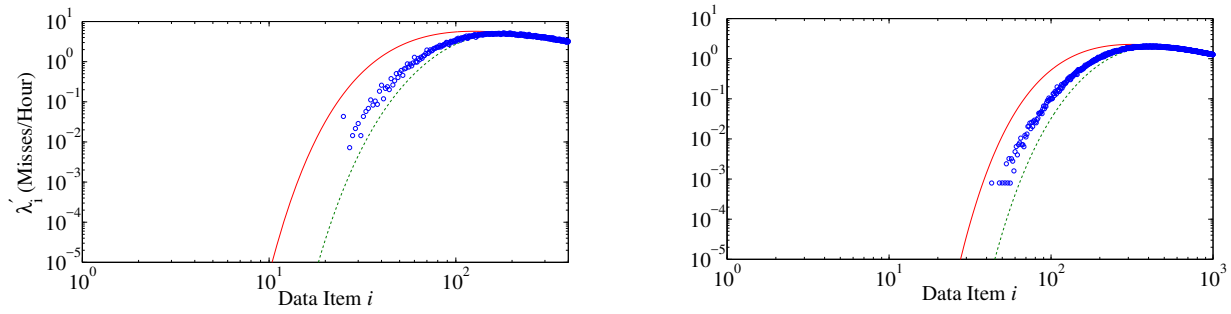


(a) $\alpha = 0.8$ **(b)** $\alpha = 1.0$

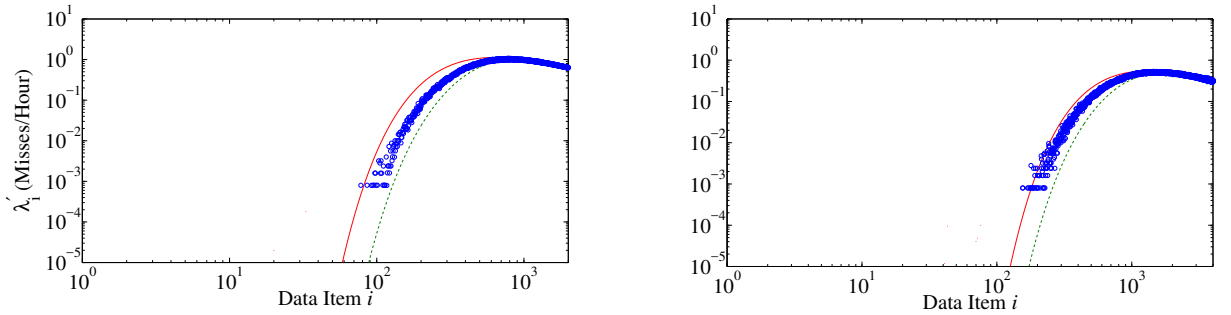


(c) $\alpha = 1.2$ **(d)** $\alpha = 1.4$

Figure 4.5: LRU-2 *vs* 2-LRU, $C = 200$, root.

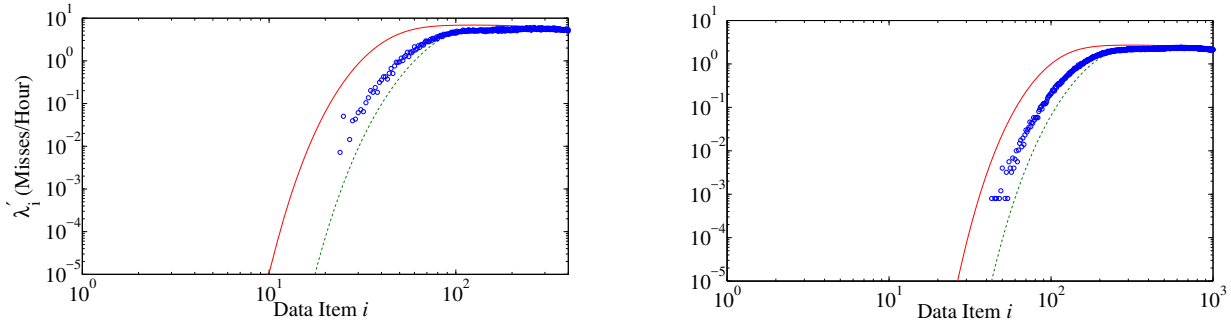


(a) $C = 200$ **(b)** $C = 500$

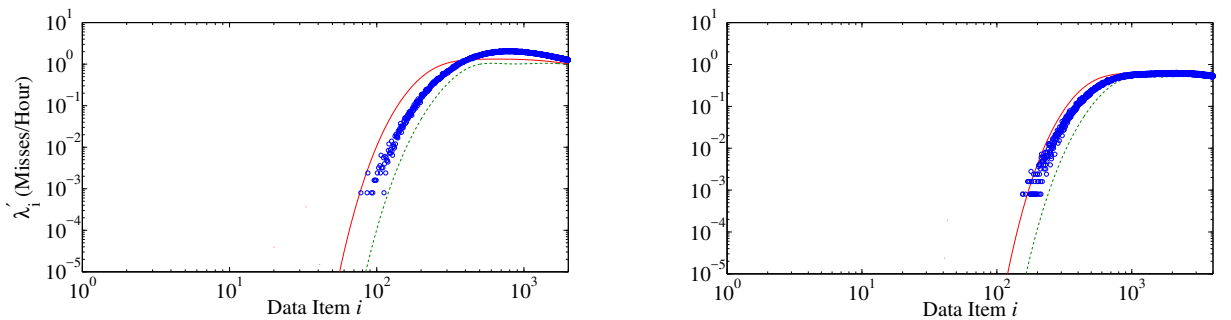


(c) $C = 1000$ **(d)** $C = 2000$

Figure 4.6: LRU-2 *vs* 2-LRU, $\alpha = 1.0$, edge.



(a) $C = 200$ **(b)** $C = 500$



(c) $C = 1000$ **(d)** $C = 2000$

Figure 4.7: LRU-2 *vs* 2-LRU, $\alpha = 1.0$, root.

of the 2-LRU algorithm with 6.72% error (absolute value of error) for $\alpha = 0.8$ and $C = 100$, that is larger than 2.62% error of Gast's model. The error of the LRU-2 model's prediction for 2-LRU however drops to 1.83% for $C = 4000$ that is smaller than 4.81% error for Gast's approximation. One can also say that the 2-LRU's hit ratio estimated by LRU-2 gets more accurate as α gets larger; i.e. for $C = 100$ for example, the error (absolute value of error) of the LRU-2's approximation for 2-LRU decreases from 5.80% to 0.44% as α moves from 0.8 to 1.4.

The error between the models' predictions and simulation results is mostly due to the violation of the IRM assumption on requests arrivals at intermediate nodes (shown in Tables 4.5-4.8), that has also been observed before by Rosensweig *et al.* [76]. The tables show that while the models approximate hit ratio at edge nodes with absolute value of error less than 2% ($|error| < 2\%$), the models have larger error to estimate the hit ratio at intermediate nodes. For $\alpha = 0.8$ for instance, $error > 20\%$ for 2-LRU and LRU-2 are observed for Dtelecom and Level3 topologies. The reason of this high error ratio for these two topologies compared to Geant and Tiger topologies is the high node degree of the nodes in DTelecom and Level3 topologies (Table 3.1). In other words, Tables 4.5-4.8 imply that the inaccuracy of models to calculate the hit ratio at an intermediate node increase by the number of children of the node.

4.4 Summary

LRU cache replacement algorithms let the data items requested only once get inserted into the cache. This results in eviction of some more popular data items from the cache. To deal with this issue, LRU-k was proposed that considers not only recency but also frequency in making decision for eviction. Because of its implementation complexity however, some other cache replacement algorithms such as LRU-2Q, ARC and k-LRU are introduced that consider both frequency and recency with low implementation overhead. In this chapter, a mathematical model for LRU-2 is proposed through extending the Che's approximation, as a specific case of LRU-k for $k = 2$. The experiments validated that the proposed LRU-2 model precisely approximates the miss rate of data items for the LRU-2 caching algorithm. The simulation results also show that although 2-LRU outperforms LRU-2 (both in individual cache and network of caches), Gast's model for 2-LRU underestimates the miss rate as either Zipf parameter (α) or cache size increases. On the other hand, the proposed model for LRU-2 calculates a better approximation of 2-LRU behaviour as either α or cache size increases. The accuracy of the models' estimations for edge and intermediate nodes are also investigated in this chapter. This investigation confirms the Rosensweig's findings about the inaccuracy of models at intermediate nodes that is caused by non-IRM arrivals of the requests [76].

Approximating the hit ratio at node u in a network of caches using Rosensweig's technique, Equations (2.17) and (2.18), needs the overall arrival rate for each data item at u and cache size of u. The overall arrival rate of data item i at u is the accumulation of arrival rates of data item i from u's neighbours. In case of a hierarchical tree of caches, the overall arrival rate of data item i at u would be the accumulation of requests

Table 4.1: Hit ratio (%), LRU-2 vs 2-LRU, Geant topology.

C	2-LRU		LRU-2		
	Sim	Model (err)	Sim	Model (err)	vs. 2-LRU Sim
			$\alpha = 0.8$		
100	9.49	9.61 (1.26%)	8.43	8.94 (6.05%)	-5.80%
500	16.89	17.31 (2.49%)	15.58	16.47 (5.71%)	-2.49%
1000	21.99	22.69 (3.18%)	20.64	21.84 (5.81%)	-0.68%
2000	29.30	30.52 (4.16%)	28.06	29.68 (5.77%)	1.30%
4000	40.72	42.77 (5.03%)	39.82	42.16 (5.88%)	3.54%
			$\alpha = 1.0$		
100	22.32	22.57 (1.12%)	20.99	21.51 (2.48%)	-3.63%
500	34.59	35.15 (1.62%)	33.13	34.11 (2.96%)	-1.39%
1000	41.62	42.42 (1.92%)	40.21	41.21 (2.49%)	-0.99%
2000	50.31	51.46 (2.29%)	49.12	50.47 (2.75%)	0.32%
4000	61.57	63.21 (2.66%)	60.82	62.23 (2.32%)	1.07%
			$\alpha = 1.2$		
100	44.21	44.67 (1.04%)	42.87	43.68 (1.89%)	-1.20%
500	60.80	61.42 (1.02%)	59.63	60.45 (1.38%)	-0.58%
1000	68.17	68.86 (1.01%)	67.17	67.27 (0.15%)	-1.32%
2000	75.61	76.41 (1.06%)	74.88	75.33 (0.60%)	-0.37%
4000	83.18	84.12 (1.13%)	82.80	83.14 (0.41%)	-0.05%
			$\alpha = 1.4$		
100	68.69	69.24 (0.80%)	67.75	68.39 (0.94%)	-0.44%
500	83.42	83.84 (0.50%)	82.85	82.54 (-0.37%)	-1.05%
1000	88.11	88.50 (0.44%)	87.70	87.69 (-0.01%)	-0.48%
2000	91.90	92.30 (0.44%)	91.64	91.76 (0.13%)	-0.15%
4000	94.93	95.38 (0.47%)	94.83	95.04 (0.22%)	0.12%

Table 4.2: Hit ratio (%), LRU-2 vs 2-LRU, Tiger topology.

C	2-LRU		LRU-2		
	Sim	Model (err)	Sim	Model (err)	vs. 2-LRU Sim
			$\alpha = 0.8$		
100	10.21	10.28 (0.69%)	9.12	9.55 (4.71%)	-6.46%
500	18.06	18.38 (1.77%)	16.72	17.46 (4.43%)	-3.32%
1000	23.36	23.94 (2.48%)	22.00	22.99 (4.50%)	-1.58%
2000	30.84	31.86 (3.31%)	29.61	30.94 (4.49%)	0.32%
4000	42.27	43.98 (4.05%)	41.39	43.01 (3.91%)	1.75%
			$\alpha = 1.0$		
100	23.86	24.07 (0.88%)	22.53	22.95 (1.86%)	-3.81%
500	36.57	37.06 (1.34%)	35.13	35.94 (2.31%)	-1.72%
1000	43.69	44.38 (1.58%)	42.32	43.20 (2.08%)	-1.12%
2000	52.33	53.32 (1.89%)	51.18	52.28 (2.15%)	-0.10%
4000	63.27	64.67 (2.21%)	62.56	63.61 (1.68%)	0.54%
			$\alpha = 1.2$		
100	46.42	46.83 (0.88%)	45.12	45.78 (1.46%)	-1.38%
500	62.88	63.43 (0.87%)	61.79	62.37 (0.94%)	-0.81%
1000	70.01	70.63 (0.89%)	69.09	69.03 (-0.09%)	-1.40%
2000	77.12	77.82 (0.91%)	76.44	76.55 (0.14%)	-0.74%
4000	84.21	87.38 (3.76%)	83.86	84.20 (0.41%)	-0.01%
			$\alpha = 1.4$		
100	70.56	71.06 (0.71%)	69.69	69.90 (0.30%)	-0.94%
500	84.60	84.97 (0.44%)	84.09	83.64 (-0.54%)	-1.13%
1000	88.99	89.32 (0.37%)	88.61	88.57 (-0.05%)	-0.47%
2000	92.50	92.85 (0.38%)	92.27	92.37 (0.11%)	-0.14%
4000	95.28	95.69 (0.43%)	95.19	95.38 (0.20%)	0.10%

Table 4.3: Hit ratio (%), LRU-2 vs 2-LRU, Dtelecom topology.

C	2-LRU		LRU-2		
	Sim	Model (err)	Sim	Model (err)	vs. 2-LRU Sim
$\alpha = 0.8$					
100	11.46	11.76 (2.62%)	10.12	10.69 (5.63%)	-6.72%
500	19.97	20.66 (3.46%)	18.38	19.30 (5.01%)	-3.36%
1000	25.63	26.63 (3.90%)	24.04	25.22 (4.91%)	-1.60%
2000	33.50	34.99 (4.45%)	32.08	33.50 (4.43%)	0.00%
4000	45.29	47.47 (4.81%)	44.28	46.12 (4.16%)	1.83%
$\alpha = 1.0$					
100	26.43	26.90 (1.78%)	24.81	25.52 (2.86%)	-3.44%
500	39.75	40.53 (1.96%)	38.11	39.01 (2.36%)	-1.86%
1000	47.03	48.02 (2.11%)	45.52	46.51 (2.17%)	-1.11%
2000	55.71	56.98 (2.28%)	54.48	55.59 (2.04%)	-0.22%
4000	66.44	68.08 (2.47%)	65.71	66.76 (1.60%)	0.48%
$\alpha = 1.2$					
100	49.92	50.55 (1.26%)	48.44	49.05 (1.26%)	-1.74%
500	66.09	66.79 (1.06%)	64.93	65.56 (0.97%)	-0.80%
1000	72.90	73.63 (1.00%)	71.95	72.24 (0.40%	-0.91%
2000	79.53	80.34 (1.02%)	78.87	78.70 (-0.22%)	-1.04%
4000	86.02	86.95 (1.08%)	85.72	85.92 (0.23%)	-0.12%
$\alpha = 1.4$					
100	73.48	74.14 (0.90%)	72.56	71.81 (-1.03%)	-2.27%
500	86.40	86.81 (0.47%)	85.89	85.04 (-0.99%)	-1.57%
1000	90.32	90.71 (0.43%)	89.97	89.56 (-0.46%)	-0.84%
2000	93.41	93.83 (0.45%)	93.22	93.36 (0.15%)	-0.05%
4000	95.80	96.33 (0.55%)	95.77	95.83 (0.06%)	0.03%

Table 4.4: Hit ratio (%), LRU-2 vs 2-LRU, Level3 topology.

C	2-LRU		LRU-2		
	Sim	Model (err)	Sim	Model (err)	vs. 2-LRU Sim
			$\alpha = 0.8$		
100	9.83	10.20 (3.76%)	8.53	9.35 (9.61%)	-4.88%
500	17.32	18.28 (5.54%)	15.77	17.18 (8.94%)	-0.81%
1000	22.48	23.90 (6.32%)	20.89	22.74 (8.86%)	1.16%
2000	29.88	32.06 (7.30%)	28.39	30.82 (8.56%)	3.15%
4000	41.47	44.78 (7.98%)	40.33	42.98 (6.57%)	3.64%
			$\alpha = 1.0$		
100	23.09	23.69 (2.60%)	21.44	22.55 (5.18%)	-2.34%
500	35.52	36.61 (3.07%)	33.79	35.27 (4.38%)	-0.70%
1000	42.61	44.03 (3.33%)	40.97	42.67 (4.15%)	0.14%
2000	51.37	53.25 (3.66%)	49.97	51.97 (4.00%)	1.17%
4000	62.68	65.16 (3.96%)	61.79	63.95 (3.50%)	2.03%
			$\alpha = 1.2$		
100	45.39	46.23 (1.85%)	43.75	44.94 (2.72%)	-0.99%
500	61.91	62.91 (1.62%)	60.55	61.72 (1.93%)	-0.31%
1000	69.19	70.27 (1.56%)	68.05	69.01 (1.41%)	-0.26%
2000	76.50	77.70 (1.57%)	75.67	76.33 (0.87%)	-0.22%
4000	83.88	85.24 (1.62%)	83.46	84.16 (0.84%)	0.33%
			$\alpha = 1.4$		
100	69.80	70.69 (1.28%)	68.67	69.58 (1.33%)	-0.32%
500	84.13	84.74 (0.73%)	83.47	82.58 (-1.07%)	-1.84%
1000	88.64	89.20 (0.63%)	88.18	88.11 (-0.08%)	-0.60%
2000	92.25	92.83 (0.63%)	91.98	92.41 (0.47%)	0.17%
4000	95.10	95.76 (0.69%)	95.02	95.43 (0.43%)	0.35%

Table 4.5: Accuracy of models, Geant topology.

C	2-LRU				LRU-2			
	Edge		Intermediate		Edge		Intermediate	
	Sim	Model (err)	Sim	Model (err)	Sim	Model (err)	Sim	Model (err)
$\alpha = 0.8$								
100	23.21	23.43 (0.95%)	7.01	7.09 (1.14%)	21.13	21.11 (-0.09%)	6.19	6.78 (9.53%)
500	36.86	37.28 (1.14%)	12.65	13.04 (3.08%)	34.81	34.80 (-0.03%)	11.61	12.64 (8.87%)
1000	44.23	44.79 (1.27%)	16.75	17.43 (4.06%)	42.44	42.46 (0.05%)	15.64	17.04 (8.95%)
2000	52.84	53.58 (1.40%)	22.95	24.19 (5.40%)	51.53	50.84 (-1.34%)	21.87	24.01 (9.79%)
4000	63.16	64.06 (1.42%)	33.37	35.61 (6.71%)	62.53	62.46 (-0.11%)	32.48	35.44 (9.11%)
$\alpha = 1.0$								
100	47.06	47.40 (0.72%)	16.37	16.57 (1.22%)	45.47	45.48 (0.02%)	15.24	15.90 (4.33%)
500	62.08	62.55 (0.76%)	26.22	26.73 (1.95%)	60.86	60.90 (0.07%)	24.88	26.06 (4.74%)
1000	68.54	69.07 (0.77%)	32.36	33.14 (2.41%)	67.57	67.63 (0.09%)	31.00	32.24 (4.00%)
2000	75.06	75.64 (0.77%)	40.50	41.72 (3.01%)	74.39	74.47 (0.11%)	39.28	40.99 (4.35%)
4000	81.68	82.34 (0.81%)	52.11	54.03 (3.68%)	81.39	81.09 (-0.37%)	51.25	53.36 (4.12%)
$\alpha = 1.2$								
100	71.64	71.98 (0.47%)	34.27	34.70 (1.25%)	70.82	70.75 (-0.10%)	32.94	33.99 (3.19%)
500	83.18	83.49 (0.37%)	50.21	50.88 (1.33%)	82.70	82.72 (0.02%)	48.89	49.98 (2.23%)
1000	87.10	87.41 (0.36%)	58.16	58.97 (1.39%)	86.75	86.52 (-0.27%)	56.95	57.31 (0.63%)
2000	90.51	90.83 (0.35%)	66.85	67.84 (1.48%)	90.29	89.58 (-0.79%)	65.89	67.02 (1.71%)
4000	93.47	93.84 (0.40%)	76.46	77.68 (1.60%)	93.39	92.93 (-0.49%)	75.91	76.76 (1.12%)
$\alpha = 1.4$								
100	87.53	87.83 (0.34%)	58.64	59.25 (1.04%)	87.22	87.17 (-0.06%)	57.49	58.43 (1.64%)
500	94.14	94.29 (0.16%)	76.35	76.90 (0.72%)	94.00	92.63 (-1.46%)	75.55	75.98 (0.57%)
1000	95.89	96.04 (0.16%)	82.62	83.15 (0.64%)	95.80	95.01 (-0.82%)	82.01	82.56 (0.67%)
2000	97.19	97.37 (0.19%)	87.98	88.51 (0.60%)	97.14	97.16 (0.02%)	87.58	87.75 (0.19%)
4000	98.12	98.38 (0.26%)	92.46	93.04 (0.63%)	98.12	98.18 (0.06%)	92.29	92.60 (0.34%)

Table 4.6: Accuracy of models, Tiger topology.

C	2-LRU				LRU-2			
	Edge		Intermediate		Edge		Intermediate	
	Sim	Model (err)	Sim	Model (err)	Sim	Model (err)	Sim	Model (err)
$\alpha = 0.8$								
100	23.21	23.43 (0.95%)	7.59	7.62 (0.40%)	21.12	21.11 (-0.05%)	6.76	7.26 (7.40%)
500	36.85	37.28 (1.17%)	13.65	13.91 (1.90%)	34.79	34.80 (0.03%)	12.58	13.46 (7.00%)
1000	44.23	44.79 (1.27%)	17.96	18.49 (2.95%)	42.43	42.46 (0.07%)	16.85	18.02 (6.94%)
2000	52.83	53.58 (1.42%)	24.39	25.39 (4.10%)	51.52	50.84 (-1.32%)	23.32	25.14 (7.80%)
4000	63.15	64.06 (1.44%)	34.96	36.78 (5.21%)	62.53	62.46 (-0.11%)	34.08	36.28 (6.46%)
$\alpha = 1.0$								
100	47.06	47.40 (0.72%)	17.72	17.85 (0.73%)	45.47	45.48 (0.02%)	16.59	17.12 (3.19%)
500	62.09	62.55 (0.74%)	28.12	28.54 (1.49%)	60.87	60.90 (0.05%)	26.80	27.78 (3.66%)
1000	68.56	69.07 (0.74%)	34.47	35.11 (1.86%)	67.58	67.63 (0.07%)	33.14	34.24 (3.32%)
2000	75.05	75.64 (0.79%)	42.72	43.74 (2.39%)	74.39	74.47 (0.11%)	41.53	42.94 (3.40%)
4000	81.67	82.34 (0.82%)	54.19	55.77 (2.92%)	81.39	81.09 (-0.37%)	53.36	55.00 (3.07%)
$\alpha = 1.2$								
100	71.63	71.98 (0.49%)	36.58	36.93 (0.96%)	70.81	70.75 (-0.08%)	35.28	36.15 (2.47%)
500	83.18	83.49 (0.37%)	52.73	53.32 (1.12%)	82.69	82.72 (0.04%)	51.48	52.29 (1.57%)
1000	87.10	87.41 (0.36%)	60.57	61.27 (1.16%)	86.74	86.52 (-0.25%)	59.44	59.53 (0.15%)
2000	90.50	90.83 (0.36%)	68.97	69.81 (1.22%)	90.28	89.58 (-0.78%)	68.07	68.72 (0.95%)
4000	93.46	93.84 (0.41%)	78.02	79.06 (1.33%)	93.38	92.93 (-0.48%)	77.51	78.38 (1.12%)
$\alpha = 1.4$								
100	87.54	87.83 (0.33%)	61.09	61.62 (0.87%)	87.23	87.17 (-0.07%)	60.02	60.38 (0.60%)
500	94.14	94.29 (0.16%)	78.14	78.62 (0.61%)	94.00	92.63 (-1.46%)	77.43	77.65 (0.28%)
1000	95.89	96.04 (0.16%)	84.03	84.48 (0.54%)	95.80	95.01 (-0.82%)	83.48	83.98 (0.6%)
2000	97.19	97.37 (0.19%)	88.97	89.43 (0.52%)	97.15	97.16 (0.01%)	88.62	88.78 (0.18%)
4000	98.12	98.38 (0.26%)	93.07	93.58 (0.55%)	98.12	98.18 (0.06%)	92.91	93.19 (0.30%)

Table 4.7: Accuracy of models, Dtelecom topology.

C	2-LRU				LRU-2			
	Edge		Intermediate		Edge		Intermediate	
	Sim	Model (err)	Sim	Model (err)	Sim	Model (err)	Sim	Model (err)
$\alpha = 0.8$								
100	23.23	23.43 (0.86%)	4.61	4.95 (7.38%)	21.11	21.11 (0.00%)	3.89	4.77 (22.62%)
500	36.85	37.28 (1.17%)	8.04	8.83 (9.83%)	34.79	34.80 (0.03%)	7.14	8.67 (21.43%)
1000	44.22	44.79 (1.29%)	10.74	11.95 (11.27%)	42.43	42.46 (0.07%)	9.76	11.8 (20.90%)
2000	52.83	53.58 (1.42%)	15.16	17.11 (12.86%)	51.51	50.84 (-1.30%)	14.11	17.54 (24.31%)
4000	63.13	64.06 (1.47%)	23.49	26.77 (13.96%)	62.51	61.40 (-1.78%)	22.35	27.17 (21.57%)
$\alpha = 1.0$								
100	47.06	47.40 (0.72%)	9.31	9.80 (5.26%)	45.48	45.50 (0.04%)	8.16	9.25 (13.36%)
500	62.09	62.55 (0.74%)	14.50	15.41 (6.28%)	60.87	60.90 (0.05%)	13.16	14.96 (13.68%)
1000	68.55	69.07 (0.76%)	18.21	19.47 (6.92%)	67.58	67.63 (0.07%)	16.80	18.95 (12.80%)
2000	75.04	75.64 (0.80%)	23.83	25.68 (7.76%)	74.39	74.47 (0.11%)	22.42	25.13 (12.09%)
4000	81.65	82.34 (0.85%)	33.57	36.45 (8.58%)	81.39	81.09 (-0.37%)	32.29	36.39 (12.70%)
$\alpha = 1.2$								
100	71.64	71.98 (0.47%)	18.42	19.17 (4.07%)	70.82	70.84 (0.03%)	16.88	17.92 (6.16%)
500	83.17	83.49 (0.38%)	28.28	29.33 (3.71%)	82.69	82.72 (0.04%)	26.58	28.42 (6.92%)
1000	87.09	87.41 (0.37%)	34.59	35.87 (3.70%)	86.75	86.52 (-0.27%)	32.89	35.02 (6.48%)
2000	90.49	90.83 (0.38%)	43.00	44.67 (3.88%)	90.29	89.58 (-0.79%)	41.45	44.31 (6.90%)
4000	93.42	93.84 (0.45%)	54.95	57.19 (4.08%)	93.38	92.93 (-0.48%)	53.81	57.21 (6.32%)
$\alpha = 1.4$								
100	87.53	87.83 (0.34%)	34.81	35.84 (2.96%)	87.21	86.83 (-0.44%)	33.07	35.47 (7.26%)
500	94.13	94.29 (0.17%)	52.85	53.95 (2.08%)	93.99	92.63 (-1.45%)	51.26	54.68 (6.67%)
1000	95.88	96.04 (0.17%)	61.87	63.03 (1.87%)	95.79	95.01 (-0.81%)	60.50	62.98 (4.1%)
2000	97.16	97.37 (0.22%)	71.25	72.52 (1.78%)	97.13	97.16 (0.03%)	70.22	70.98 (1.08%)
4000	98.07	98.38 (0.32%)	80.67	82.23 (1.93%)	98.11	98.63 (0.53%)	80.19	82.06 (2.33%)

Table 4.8: Accuracy of models, Level3 topology.

C	2-LRU				LRU-2			
	Edge		Intermediate		Edge		Intermediate	
	Sim	Model (err)	Sim	Model (err)	Sim	Model (err)	Sim	Model (err)
$\alpha = 0.8$								
100	23.22	23.43 (0.90%)	3.86	4.29 (11.14%)	21.11	21.11 (0.00%)	3.09	4.24 (37.22%)
500	36.85	37.28 (1.17%)	6.67	7.80 (16.94%)	34.79	34.80 (0.03%)	5.75	7.79 (35.48%)
1000	44.23	44.79 (1.27%)	8.93	10.69 (19.71%)	42.43	42.46 (0.07%)	7.94	10.71 (34.89%)
2000	52.83	53.58 (1.42%)	12.74	15.59 (22.37%)	51.51	50.84 (-1.30%)	11.65	16.11 (38.28%)
4000	63.14	64.06 (1.46%)	20.15	25.03 (24.22%)	62.51	63.82 (2.10%)	18.97	25.74 (35.69%)
$\alpha = 1.0$								
100	47.06	47.40 (0.72%)	7.70	8.36 (8.57%)	45.48	45.48 (0.00%)	6.54	8.23 (25.84%)
500	62.09	62.55 (0.74%)	12.03	13.33 (10.81%)	60.87	60.90 (0.05%)	10.70	13.14 (22.8%)
1000	68.55	69.07 (0.76%)	15.20	17.04 (12.11%)	67.58	67.63 (0.07%)	13.80	16.80 (21.74%)
2000	75.04	75.64 (0.80%)	20.16	22.89 (13.54%)	74.39	74.47 (0.11%)	18.72	22.62 (20.83%)
4000	81.67	82.34 (0.82%)	29.07	33.45 (15.07%)	81.40	81.09 (-0.38%)	27.73	33.62 (21.24%)
$\alpha = 1.2$								
100	71.63	71.98 (0.49%)	15.38	16.40 (6.63%)	70.81	70.75 (-0.08%)	13.81	16.12 (16.73%)
500	83.17	83.49 (0.38%)	24.03	25.57 (6.41%)	82.69	82.72 (0.04%)	22.32	24.97 (11.87%)
1000	87.09	87.41 (0.37%)	29.79	31.73 (6.51%)	86.75	86.52 (-0.27%)	28.04	31.32 (11.70%)
2000	90.49	90.83 (0.38%)	37.76	40.30 (6.73%)	90.28	89.58 (-0.78%)	36.09	40.67 (12.69%)
4000	93.43	93.84 (0.44%)	49.62	53.15 (7.11%)	93.38	93.03 (-0.37%)	48.31	58.58 (21.26%)
$\alpha = 1.4$								
100	87.53	87.83 (0.34%)	29.99	31.41 (4.73%)	87.21	87.17 (-0.05%)	28.17	30.74 (9.12%)
500	94.13	94.29 (0.17%)	47.36	49.01 (3.48%)	93.99	92.63 (-1.45%)	45.61	48.86 (7.13%)
1000	95.88	96.04 (0.17%)	56.59	58.36 (3.13%)	95.80	95.01 (-0.82%)	55.01	58.77 (6.84%)
2000	97.17	97.37 (0.21%)	66.06	68.55 (2.93%)	97.13	97.16 (0.03%)	65.35	67.56 (3.38%)
4000	98.08	98.38 (0.31%)	77.14	79.46 (3.01%)	98.11	98.18 (0.07%)	76.48	78.68 (2.88%)

for data item i coming from u's children. Although findings of this chapter are based on the four publicly available ISP topologies, the proposed LRU-2 model could be applied to any arbitrary topology. Studying the performance of LRU-2 and the accuracy of the proposed LRU-2 model for arbitrary topologies could be part of future work.

Chapter 5

Geographical Locality in Users' Requests

5.1 Motivation

As overviewed in Section 2.4, most studies investigate the network of caches under IRM where the temporal and geographical locality in users' requests are ignored. As discussed in Section 2.4, temporal locality is well studied and modelled in the literature [45, 90, 89]. On the other hand, most studies use simplifying assumptions for the geographical locality in users' requests, such as identical request patterns, in absence of any trace-driven methodologies since ICNs are not yet deployed. As mentioned in Section 2.1, requests for Internet services have been modelled as Zipf distributions for many years [30, 44]. There are two aspects of the request distribution that have been studied independently and collectively: global request patterns (e.g. the overall number of views of YouTube videos) [23, 82] and local request patterns (e.g. the number of views of YouTube videos in a geographical region at the edge of the Internet such as a university campus) [47, 61, 100]. The presence of these Zipf distributions has enabled effective caching policies and architectures (such as LRU) to reduce disk and network traffic and improve response times for clients/users.

Analysis of YouTube video requests on a university campus by Gill *et al.* shows that more than 68.1% of the videos were only requested once [47]. In addition, some other studies show that the local users' requests for media data items follows a Zipf distribution [100]. Other studies have also shown that the global popularity of YouTube videos also follows a Zipf distribution [23, 82]. The evidence suggests that there are scenarios in which the distribution of both local and global requests follow Zipf distributions. However, the findings of Zink *et al.* suggest that there is a very small correlation between local and global popularity of the videos in YouTube [100]. Correlation between regions is shown to vary with respect to geographic location (country) and/or other demographic qualities [90].

The goal of this chapter is threefold: first, an algorithm is developed in Section 5.2 that can generate synthetic traffic for regional caches that possesses Zipf properties as well as produces a global Zipf distribution. Varying parameter settings will allow different shapes/scales of the global distribution. It will be useful in sensitivity analysis for all multi-level caching architectures. Second, *local search* is provided to the ICN networks in Section 5.3. Next, experimentation suitable for a hierarchical cache typical of the deployment in an ICN provided in Section 5.4. Finally, Section 5.5 summarizes the chapter and gives a summary of limitations for this chapter.

5.2 Traffic Generator Principles

Here, the algorithm to generate local distributions with Zipf properties out of a global Zipf distribution is proposed in Section 5.2.1. The distribution of users' requests at a local region represents the pattern of users' requests in a geographical region at the edge of the Internet such as the users' requests for YouTube videos at a university campus, a city or a province. Geographically local regions have different scales (i.e. number of users) that result in different overall request rates for each region (i.e. users in a city generate more traffic compared to the users in a university campus). The accumulation of all users' requests (e.g. for YouTube videos) from all local regions represents the global distribution. The algorithm generates distributions of users' requests for a set of geographically local regions with random scales. The properties of generated local distributions are then studied in Section 5.2.2. Section 5.2.3 explains the output of the proposed algorithm in case of existing neighbouring regions with similar distributions.

5.2.1 Weighted Random Regional Request Distributions

Having notations in Table 3.2 and assuming $\Gamma_r = \{1, 2, \ldots, N\}$, the popularity of data items in region r for users' requests follows a Zipf distribution with parameter α_r (i.e. $p_{r,i} \propto (1/\Pi_{r,i})^{\alpha_r}$). Consequently, the request rate for data item i in r is calculated as $\lambda_{r,i} = p_{r,i}\lambda_r$. Then, assuming two subregions u and v, the order of data items' popularity in either u or v is different from the order in enclosing region r and complies with the following premises:

$$\Pi_u \neq \Pi_r, \tag{5.1}$$

$$\Pi_v \neq \Pi_r, \tag{5.2}$$

$$\Pi_u \neq \Pi_v, \tag{5.3}$$

$$\Gamma_r = \Gamma_u = \Gamma_v, \tag{5.4}$$

$$\lambda_u + \lambda_v = \lambda_r. \tag{5.5}$$

Premises (5.1)-(5.3) express that the order of data items in u, v and r should be different. The first one, for example, means that the order of items in u should be different from the order of items in r. These premises imply that the popularity of data items in different regions are not equal as the characteristic of geographical locality. Equation (5.4) means that all regions should include all the available data items. Equation (5.5) implies that the total request rate of requests in r is divided between u and v. In (5.1) to (5.5), Π_u, Π_v, α_u, α_v, λ_u and λ_v are not known. Algorithm 5.1 divides the global set of data items' popularity into two subsets, each one providing a Zipf distribution.

The algorithm uses S_r^{λ}, the minimum/maximum value of Zipf parameter (α_-/α^-), threshold t and ψ as input and returns the unknowns in (5.1) to (5.5). Algorithm 5.1 proceeds iteratively, evaluating the correspondence with a global Zipf distribution and low correlation between the component regional distributions.

Algorithm 5.1 Pseudo-Bisect Zipf distribution

procedure DIVIDE–ZIPF
  INPUT:
    $\lambda_r^s, \alpha_-, \alpha^-, t, \psi$
  OUTPUT:
    Π_v, Π_u, α_v, α_u, S_v^λ *and* S_u^λ
  while *true* **do**
    $T \leftarrow \{1, 2 \dots N\}$
    $i \leftarrow Weight\text{-}Random(T, \psi)$
    $T \leftarrow T - \{\Pi_{r,i}\}$
    $\Pi_{u,i} \leftarrow 1$
    Randomly pick $\lambda_{u,i}$, $0 < \lambda_{u,i} < \lambda_{r,i}$
    $\lambda_{v,i} \leftarrow \lambda_{r,i} - \lambda_{u,i}$
    Pick up random α_u, $\alpha_- < \alpha_u < \alpha^-$
    $\theta \leftarrow \sum_{k=1}^{N} (1/k)^{\alpha_u}$
    $p_{u,i} \leftarrow 1/\theta$
    $\lambda_u \leftarrow \lambda_{u,i}/p_{u,i}$
    $\lambda_v \leftarrow \lambda_r - \lambda_u$
    for $j = 2$ *to* N **do**
      $\Lambda_{u,j} \leftarrow \lambda_u\big((1/j)^{\alpha_u}/\theta\big)$
    end for
    $j \leftarrow 2$
    while $T \neq \emptyset$ **do**
      $k \leftarrow Weight\_Random(T, \psi)$
      if $\Lambda_{u,j} \leq \lambda_{r,k}$ **then**
        $\Pi_{u,k} \leftarrow j$
        $\lambda_{u,k} \leftarrow \Lambda_{u,j}$
        $T \leftarrow T - \{j\}$
        $\lambda_{v,k} \leftarrow \lambda_{r,k} - \lambda_{u,k}$
        $j \leftarrow j + 1$
      end if
    end while
    if *Is-Zipf*(S_v^λ) **then**
      break
    end if
  end while
end procedure

The algorithm starts with weighted random selection of a data item in region r, (line 8). The *Weight_Random* function is described in Algorithm 5.3. The selected item is assumed to be the most popular data item in region u (line 10). Then, the algorithm assigns a portion of $\lambda_{r,i}$ as the request rate for i in region u, $\lambda_{u,i}$ (line 11). The remaining request rate for i, is assigned as the request rate for the item in region v (line 12). Having the request rate for the most popular data item in region u, the algorithm then randomly selects α_u (line 13). Based on $\lambda_{u,i}$ and α_u, λ_u can be now calculated (lines 14-16). Having λ_u and α_u, the algorithm then calculates the request rate for other ranks in u (lines 18-20). In the next stage, the algorithm assigns data items to ranks in $[2, N]$ for region u (lines 21-31). In this regard, the algorithm starts with the second rank in u (line 21) since i is already assigned to the first rank in u. For the j^{th} rank in region u and among the data items with ranks in T with request rate larger than $\Lambda_{u,j}$, the algorithm randomly picks up k and considers it as the j^{th} most popular item in u (lines 23-30). Meanwhile, the request rate for item k in region v is also calculated. In the final stage, the algorithm evaluates the distribution of request rates in v to determine how close to a Zipf distribution it is (line 32).

The algorithm that finds out if a distribution is close to a Zipf distribution is depicted in Algorithm 5.2. That method repeatedly creates z as a Zipf distribution with α_z over range of $[\alpha_-, \alpha^-]$ and uses R^2 as the coefficient of determination,

$$R^2(\Lambda_z, \Lambda_v, N) = 1 - \frac{\sum_i (\Lambda_{z,i} - \Lambda_{v,i})^2}{\sum_i (\Lambda_{v,i} - \frac{\sum_j \Lambda_{v,j}}{N})^2}, \tag{5.6}$$

to compare the popularity distribution of data items in the region v with z. If the correlation is larger than t, the request distribution in region v is sufficiently close to a Zipf distribution. If the distribution of request rates in v is sufficiently close to a Zipf distribution, the algorithm ends. Otherwise, the algorithm finds a new order for the data items in u by repeating lines 6-35 of the algorithm.

The *Weight_Random* function in Algorithm 5.3 picks up the data items such that the rank of data item i in u gets closer to the rank of item i in r for smaller ψ. Equation 5.9 depicts the set T that is used in Algorithm 5.3; T is a sorted set of ranks in u which no data item is yet assigned to in the process of generating distribution for region u. Algorithm 5.3 picks up a corresponding data item from the ranks in T for the j^{th} most popular rank in region u. Having a smaller set of left ranks for most popular data items in r, shown by T_j in (5.9), helps that the next item picked up for region u (lines 8 and 23 in Algorithm 5.1) has a similar rank in r. T_j is a subset of T such that

$$\frac{1}{\lambda_r} \sum_{\forall t_i \in T_j} \Lambda_{r,t_i} < \psi, \tag{5.7}$$

and

$$\frac{1}{\lambda_r} \sum_{\forall t_i \in T_{j+1}} \Lambda_{r,t_i} > \psi, \tag{5.8}$$

in which $0 \leq \psi \leq 1$. A smaller ψ ends in a smaller T_j that consequently results in generating distribution for region u that is more similar to the distribution in region r.

$$T = \{\underbrace{t_1, t_2, t_3, t_4, \ldots, t_j}_{T_j}, t_{j+1}, \ldots\}, t_i < t_j \forall i < k \tag{5.9}$$

Algorithm 5.2 Is a distribution Zipf

```
1: procedure IS–ZIPF
2:     INPUT:
3:         S_r^λ, λ_r, t, N, α_−, α^−
4:     OUTPUT:
5:         c
6:     DEFINE:
7:     c ← false
8:     max_c ← 0
9:     find corresponding Λ_{r,j} from S_r^λ, 1 ≤ j ≤ N
10:    for α = α_− to α^− do
11:        g ← 0
12:        for j = 1 to N do
13:            g ← g + 1/j^α
14:        end for
15:        for j = 1 to N do
16:            Λ_{z,j} = (λ_r/g) 1/j^α
17:        end for
18:        rs ← R^2(Λ_r, Λ_z, N) , (5.6)
19:        if max_c < rs then
20:            max_c ← rs
21:        end if
22:    end for
23:    if max_c < t then
24:        c ← true
25:    end if
26: end procedure
```

To create k sub-request patterns, Algorithm 5.1 is applied to region u with the largest arrival rate λ_u as shown in Algorithm 5.4. Table 5.1 and Figure 5.1 show an example of the output of Algorithm 5.4 for 20000 data items, with $k = 10$, $\lambda_r = 40$, $\alpha_r = 1.0$, $t = 0.9$ and $\psi = 1.0$. As indicated in the table, different

Algorithm 5.3 Weighted random function

1: **procedure** WEIGHTED–RANDOM
2: **INPUT:**
3: T, ψ
4: **OUTPUT:**
5: *data item* l
6: $s \leftarrow 0$
7: *find* $T_j = \{t_1, \ldots, t_j\}$ *from* (5.7) *and* (5.8)
8: **for** $i = 1$ *to* j **do**
9: $s \leftarrow s + \Lambda_{r,t_i}$
10: **end for**
11: $wr \leftarrow rand(0, s)$
12: $x \leftarrow 0$
13: **for** $i = 1$ *to* j **do**
14: $x \leftarrow x + \Lambda_{r,t_i}$
15: **if** $wr \leq x$ **then**
16: *find* l *so that* $\Pi_{r,l} = t_i$
17: *break*
18: **end if**
19: **end for**
20: **end procedure**

subsets have different λ_u, α_u and Π_u. Figure 5.1 illustrates the rank/frequency distributions for 10 regions as well as region r in log-log scale. The generated distributions have Zipf properties with different values of α. Table 5.1 depicts the geographical locality in distributions of regions. For example, data item 15 is the most popular data item in region u_1; the contribution of users' requests in this region compose 1% of total requests. While data item 1 is the global most popular data item, Table 5.1 depicts that this data item is among the top five popular data items in all regions except u_3, u_4 and u_5. While the popularity of data items in these three regions are very different from the global region r, their contribution in shaping the overall requests is also very small (i.e. less than 0.33%). This implies that there may be some local regions with very local popular data items that contribute less than 1% to the overall requests in the global network.

The execution time of a sequential implementation of Algorithm 5.1 is examined for different values of N and α. While α has no influence on the execution time of the algorithm, the execution time of the algorithm increases proportionally with $O(N^2)$. For 200000 data items, the computation time was close to 2 hours on an Intel(R) Xeon(R) CPU E5-2690 v2 @ 3.00GHz.

Algorithm 5.4 Create k sub-request patterns with Zipf properties

1: **procedure** DIVIDE–ZIPF–TO–K
2: **INPUT:**
3: $S_r^{\lambda}, \alpha_{-}, \alpha^{-}, t, k$
4: **OUTPUT:**
5: $\Pi_1 \dots \Pi_k, \alpha_1 \dots \alpha_k, S_1^{\lambda} \dots S_k^{\lambda}$
6: $W \leftarrow \{\}$
7: **while** k subsets not found **do**
8: *find* Π_u, Π_v, α_u, α_v, S_u^{λ} *and* S_v^{λ} *as the output*
9: *of Divide-Zipf*$(S_r^{\lambda}, \alpha_{-}, \alpha^{-}, t)$
10: $W \leftarrow W \bigcup u \bigcup v$
11: *find* $u \in W$ *with the highest* λ_u
12: $r \leftarrow u$
13: **end while**
14: **end procedure**

5.2.2 Analysis of Regional Distribution Properties

To study the independence properties of the proposed subregion technique, a series of experiments with Algorithm 5.4 is performed. The parameters in Algorithm 5.4 for these experiments are set as follows: $N = \{1000, 5000, 10000, 20000\}$, $\lambda_r = 40$, $\alpha_r = \{0.8, 1.0, 1.2, 1.4\}$, $k = 22$ (chosen to match the number of regions in Geant topology, Table 3.1), $\alpha_{-} = 0.5$, $\alpha^{-} = 2.0$, $t = 0.9$, $\psi = 1.0$.

The following two correlations are taken into account as well: (1) global-region-correlation: the average

Table 5.1: A sample output of Algorithm 5.4.

u_x	λ_x	α_x	Π_x										
u_1	0.416	1.64	15	13	2	7	33	1	29	80	64	3	...
u_2	0.201	1.70	29	56	3	55	1	559	2	231	10	5	...
u_3	0.040	1.34	245	36	153	103	38	43	23	17	4238	6364	...
u_4	0.001	1.41	14867	78	2	5	630	6069	85	16693	44	257	...
u_5	0.088	1.42	76	13	3	87	525	21	1211	5	9	38	...
u_6	8.640	1.34	1	3	5	7	2	10	6	16	22	26	...
u_7	6.484	0.89	2	1	9	6	11	7	14	25	5	19	...
u_8	11.820	0.98	2	1	6	8	9	12	11	4	14	17	...
u_9	3.078	0.90	4	2	1	5	7	8	3	46	27	19	...
u_{10}	9.234	0.90	4	2	1	5	3	24	7	6	16	8	...
u_r	40	1.0	1	2	3	4	5	6	7	8	9	10	...

correlation between the global request distribution and all traffic distributions of all regions; (2) pairwise-region-correlation: the average correlation between the request distribution of all regions. The distribution of each of these two correlations against σ_α, the variation of α_u between the regions, and σ_λ, the variation of λ_u over the regions, is studied in Figures 5.2 and 5.3 respectively. To have distributions for correlations over a relatively large range of σ_α and σ_λ, Algorithm 5.4 ran 100 times for each permutation of α_r and N values. The output of each run gives us $\Pi_1 \dots \Pi_{22}, \alpha_1 \dots \alpha_{22}$ and $\lambda_1 \dots \lambda_{22}$.

Figure 5.2 shows the correlations based on σ_α. One can say that σ_α has no influence on the correlations. This means that region u and v may have similar Zipf parameter α while having a very small correlation that is caused by different order of ranks in the two regions. It is also worthy to note that the global-region-correlation is substantially higher than the pairwise-region-correlations. This is somewhat to be expected as the most popular items in a subregion are chosen from the most popular items in the global region, or the remaining largest region. This would lead to some amount of correlation. The low pairwise correlation shows the independence between regions.

Figure 5.3 on the other hand, depicts how correlations (1) and (2) change over σ_λ for a constant value of $\alpha = 1.0$. A larger σ_λ results in smaller pairwise-region-correlation and global-region-correlation. Assuming u and v as two sub distributions and a large variance between λ_u and λ_v, the findings of Figure 5.3 show that there is a small correlation between u and v. One reason is that if λ_u is much larger than λ_v, Algorithm 5.1 probably have chosen most popular data items in region u from the most popular data item in region r. On the other hand, for small λ_v, Algorithm 5.1 have chosen most popular data items in region v not necessarily from the most popular data item in region r. This results in different ranks of data items in the two regions that makes the correlation between two regions smaller. Similar behaviour for both correlations is observed from the generation of sub-distributions for values of $\alpha_r = \{0.8, 1.2, 1.4\}$.

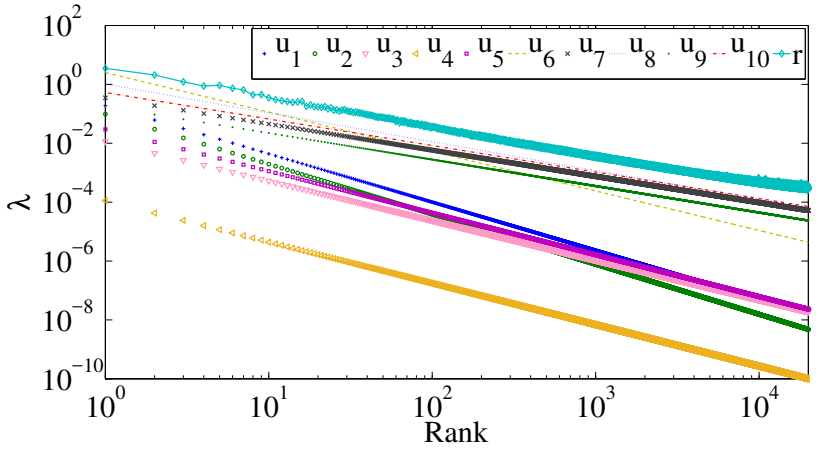


Figure 5.1: Result of Algorithm 5.1.

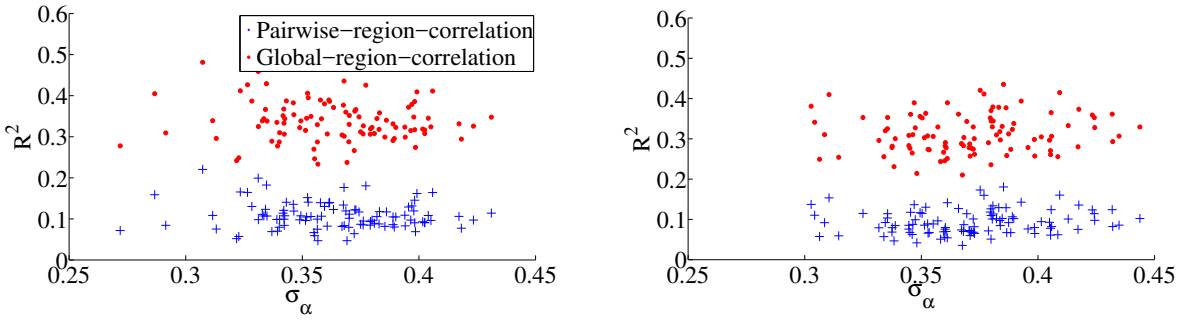
Figure 5.4 shows the global-region-correlation for different values of N and α. The correlation decreases as N increases, since line 23 in Algorithm 5.1 uses a weighted random function to find an order for data items in region u that is different from the order of data items in the original distribution. Larger population sizes result in more permutations of data items helping the algorithm to produce sub distributions with alternate orders of data items from the original distribution.

Figure 5.4 also demonstrates that the correlation increases as α increases. A steeper slope (higher α) for the distribution leaves fewer popular data items as possible choices for the most popular data items in sub distribution u (i.e a higher popularity *decay*). This results in sub traffic distributions with a similar order of popular data items. The same patterns are found in Figure 5.5 for the pairwise-region-correlation for different values of N and α.

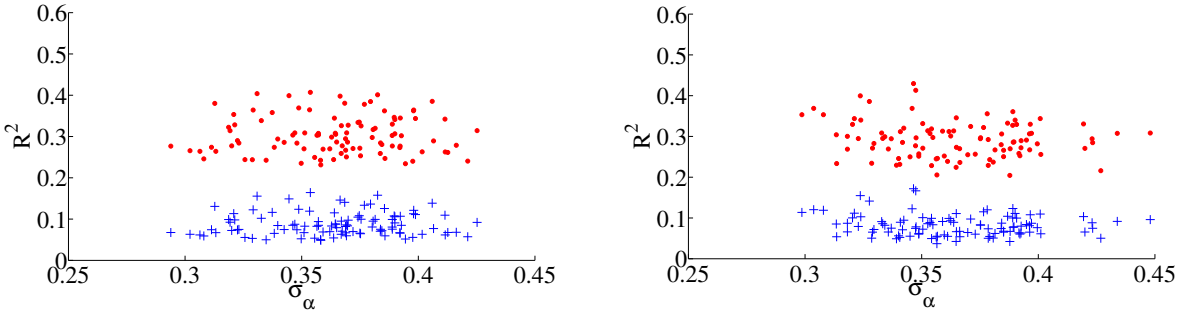
5.2.3 Neighbouring Regions with Similar Distributions

While there is small correlation between the distribution of geographically local regions and the global region, distribution of users' requests in the same city and/or country have shown to have more correlated traffic [90]. This means, if item i is popular in region u, it is probably popular in neighbouring region v as well. Having similar distributions for neighbouring regions u and v in Algorithm 5.1 is achieved through parameter ψ in weighted random Algorithm 5.3.

To evaluate the performance of Algorithm 5.3, the average similarity between the distribution of users' requests for different values of ψ and n is studied; n depicts the percentage of regions in the network that have similar distributions. In this regard, the settings in the first paragraph of previous section are used. Assume $\Pi^{-1}_{r,j}$ is the inverse function of $\Pi_{r,i}$ that shows the id of j^{th} most popular data item in distribution r. Having vectors $U = [\Pi^{-1}_{u,1}, \Pi^{-1}_{u,2}, \dots \Pi^{-1}_{u,N}]$ and $V = [\Pi^{-1}_{v,1}, \Pi^{-1}_{v,2}, \dots \Pi^{-1}_{v,N}]$ as the id of data items sorted based on their popularities in regions u and v respectively, the similarity between these two vectors is calculated

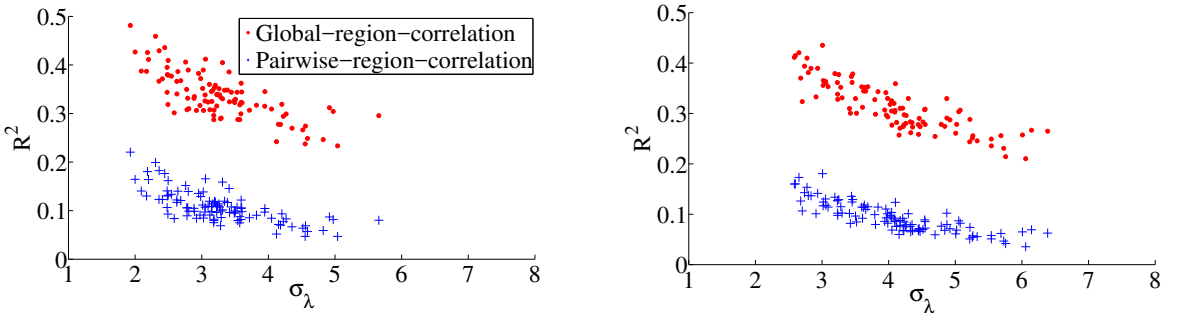


(a) $N = 1000$ **(b)** $N = 5000$

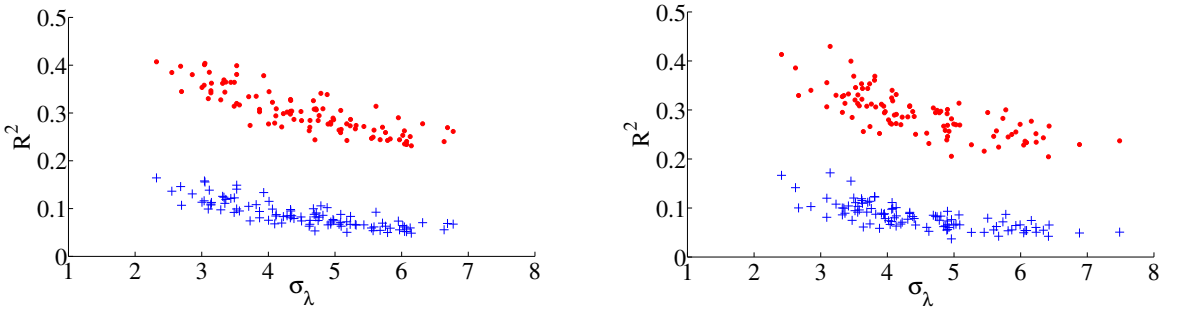


(c) $N = 10000$ **(d)** $N = 20000$

Figure 5.2: Correlations of request patterns vs. σ_α; $\alpha_r = 1.0$.



(a) $N = 1000$ **(b)** $N = 5000$



(c) $N = 10000$ **(d)** $N = 20000$

Figure 5.3: Correlations of request patterns vs. σ_λ; $\alpha_r = 1.0$.

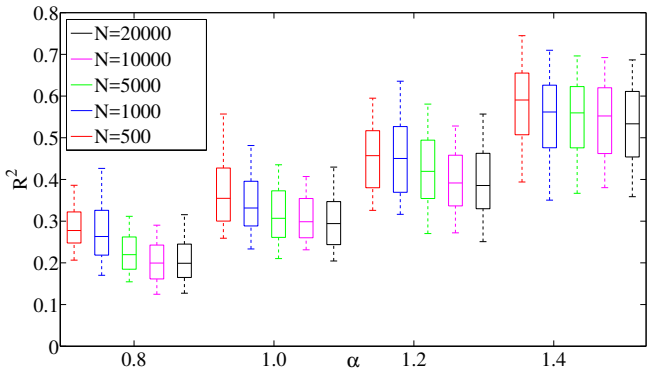


Figure 5.4: Global-region-correlation vs. α and N.

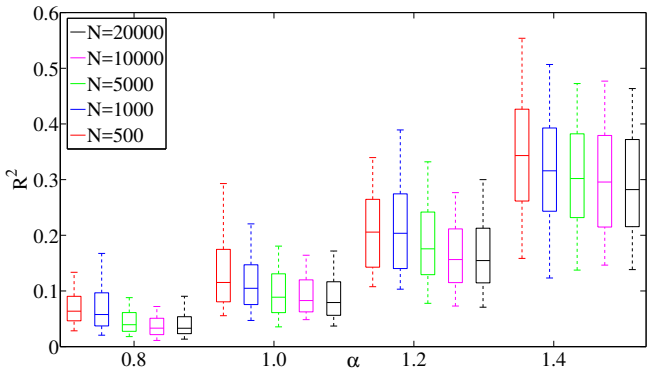


Figure 5.5: Pairwise-region-correlation vs. α and N.

through the following Normalized Squared Euclidean distance:

$$0.5 \times \frac{Var(U-V)}{Var(U)+Var(V)}. \tag{5.10}$$

Figure 5.6 shows the average increased similarity ratio between regions as a function of ψ. As ψ increases, the average similarity decreases since larger ψ results in a larger subset T_j. A larger n results in an increase in the average similarity. Having $\psi = 0.1$ and $n = 10\%$ ends in 6% increased average similarity ratio. For $\psi = 0.1$ and $n = 40\%$ however, average increased similarity ratio jumps to 30%. This figure also shows that an increase in n has a larger influence on average increased similarity ratio, and an increase in ψ has less influence on the average increased similarity ratio.

Figure 5.7 shows the correlation between regions as a function of ψ for different n values. This figure shows that although the similar distributions for neighbouring regions increases the correlation, the correlation is still small and less than 0.18.

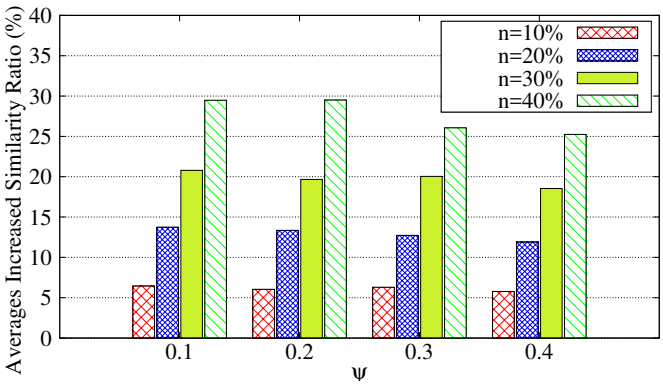


Figure 5.6: Averages increased similarity ratio between distributions as a function of ψ.

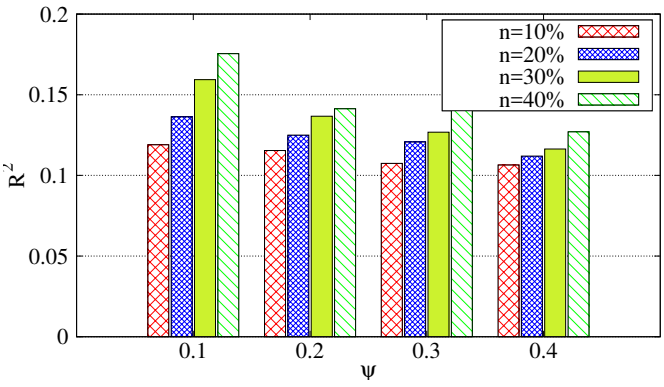


Figure 5.7: Correlation between distributions as a function of ψ.

5.3 Local search

In standard discovery/delivery mechanism of ICNs using an on-path caching mechanism (combination of content discovery/delivery and caching mechanisms explained in Chapter 1), a request for data item i from a user connected to ICN node u is forwarded by ICN nodes from u to the server on a shortest path from u to the server (Figure 1.1). If the data item is found in the cache of any of the nodes on this path, a copy of the data item is then sent to the user. Alongside this standard discovery/delivery mechanism, a local search among u's neighbouring ICNs that are not on the shortest path from u to the server may result in discovering i. In other words, deploying a local search among the ICN node's neighbours before running the standard discovery/delivery mechanism may discover data items closer to u. The probability of discovering data item i in u's neighbouring ICN v depends on the popularity of i at v. If i is a popular data item at v, then, there is a good chance that i is cached at ICN node v. Therefore, a local search for data item i from ICN node u will be a hit at ICN node v. For example, Lu *et al.* provide users with caching capabilities so that users can share their caches among themselves [60]. They propose a lookup mechanism in which requests are forwarded

to neighbouring users.

In this section, support for *local search* lookup is added to the standard content discovery/delivery mechanism in ICNs (explained in Chapter 1). Edge ICN node e, will launch a local search for data item i before the standard content discovery/delivery mechanism. ICN node e broadcasts a local search message for i to its neighbouring ICN nodes (including its parent). ICN node e also sets a timer for item i after broadcasting the local search message for i. The message contains the *nsd* (stands for neighbourhood search depth), that determines how long a local search message can travel in the network. This property works exactly like TTL. Any ICN node that receives a local search message, decreases the *nsd* of the message by one, and then broadcasts it to all its neighbouring ICN nodes except the one from which the message arrived.

To prevent an ICN node from broadcasting multiple local search messages for i coming from different ICN nodes in the network, a table called PLST, which stands for Pending Local Search Table, is introduced. Each entry in this table consists of a data item and a list of the links from which the local search messages for the corresponding data item arrived. This table works exactly like the PIT in CCN [49], although there is one PLST for each outgoing link of ICN node r. When a local search request for data item i arrives at ICN node r through link a, the ICN node checks PLST for each of its outgoing links. For the PLST of outgoing link o, depicted by PLST_o, if there is no entry for i, a new entry for i is inserted into the PLST_o, a is added to the entry, *nsd* of the local search message is decreased by one, and then the messages is sent to the ICN node which is the other side of link o. If there is an entry in PLST_o for i, a local search message for i has been previously processed. In this case, the ICN node only adds a to the entry.

If ICN node w has a copy of i, a *local search reply* message for i is sent back to the sender. ICN node w also stops broadcasting the local search message. Upon receiving a local search reply message for data item i at ICN node v from link a, ICN node v queries the PLST_a for i. Then v broadcasts the reply message to all links in the corresponding entry and deletes the entry. When edge ICN node e receives the local search reply message for i that is produced by w, e adds (w, d) to list l_i (the list of all ICN nodes in radius of *nsd* that have a copy of i); w is the ICN node that has a copy of item i, and l is the distance from e to w. When i's timer expires, ICN node e retrieves a copy from the closest ICN node w in l_i. If l_i is empty, the ICN node performs the default search.

To avoid changing the popularity distribution in neighbouring regions, when ICN node w receives a local search message, it does not treat it as a request coming from either its direct users or descendant ICN nodes. This isolation prevents local search from changing the popularity distribution, and caching performance in the neighbouring regions. For future work, there should be some consideration of weighting local vs. neighbourhood requests. There might be some value to including a local search request as part of the popularity distribution of requests to the local cache.

5.4 Performance Evaluation

In this section, the influence of users' request with geographical locality on caching performance is studied for realistic topologies Geant, Tiger, Level3 and Dtelecom (Table 3.1). This is compared with identical request patterns for different cache allocation strategies and with a simplistic local search heuristic.

5.4.1 Experimental Methodology

Each edge node in the four topologies represents a region. Furthermore, for data delivery as well as propagation of interest messages, an ICN overlay tree is constructed in which the server is connected to the root. Users are only connected to the edge regions; edge ICN nodes receive exogenous traffic while the intermediate ICN nodes receive endogenous traffic. For geographic locality, Algorithm 5.4 is used to produce k different traffic distributions. The parameter k is determined by the number of edge ICN nodes; 10, 10, 41 and 61 for Geant, Tiger, Level3 and Dtelecom topologies respectively (Table 3.1). The simulation parameters are set as follows: $N = 20000$, global Zipf shape parameter $\alpha_r = 1.0$, the global request rate $\lambda_r = 40$, $\alpha_- = 0.5$, $\alpha^- = 2.0$, $t = 0.9$ and $\psi = 1.0$. Algorithm 5.4 ran for 100 times. Out of these 100 outputs, 5 were selected such that (1) request distributions of the sub-regions do not have extreme variance in their overall arrival rates, and (2) the output results in small global-region-correlation and pairwise- region-correlation. In other words, out of 100 outputs of Algorithm 5.4, 5 with σ_λ around the middle of x axis in Figure 5.3 are selected. Therefore, in the graphs presented, the bars represent the average of 5 runs (each run has unique output of Algorithm 5.4), with error bars indicating 95% confidence intervals.

For a given total cache budget C for all ICN nodes of a topology, a number of policies are used to determine the relative sizes of each cache. The following four policies/scenarios are considered to divide the cache budget among the regions:

1. Proportional-edge-only (s_1): the edge ICN node cache size is proportional to the each region's request rate ($C_u \propto \lambda_u$ if u is an edge regions; otherwise $C_u = 0$).

2. Proportional-all-network (s_2): the cache budget is distributed among the all regions proportional to the each region's request rate. The intermediate regions only receive the requests missed in their children nodes.

3. Equal-edge-only (s_3): ($C_u = C/|E|$ if u is an edge ICN node; otherwise $C_u = 0$).

4. Equal-all-network (s_4): ($C_u = C/|R|$).

Distribution of cache budget C among the nodes in s_1, s_3 and s_4 is straightforward. Allocating cache budget C proportionally among all the nodes in s_2 is a bit tricky as the request rates at intermediate node are not available. Algorithm 5.5 distributes the cache budget C among all regions in the network according to scenario s_2. Users are only connected to the edge of the ICN tree. As a result, only the request rates at

the edge of the network are known. The request rates at one level higher than the edge of the ICN tree are composed of the miss rates at edge level. To calculate the miss rates at the edge, cache budget C is divided proportionally among the edge nodes. Based on this distribution of cache budget, the miss rates at edge nodes can be calculated. Now, the request rates at the two lowest levels of the tree are known. Therefore, the cache budget is allocated to the ICN nodes in these two levels. This technique for distributing C among all nodes in the ICN tree is applied to higher levels step by step.

This algorithm uses as input C, R and request rate for each data item i at edge regions. The algorithm also needs the structure of the overlay ICN tree. In this regard, the algorithm is fed with set of region r's child regions shown by R_r, $\forall r \in R$. The output of the algorithm will be the portion of cache budget allocated to region r, $\forall r \in R$. Having λ_r initially only for the edge ICN nodes, Algorithm 5.5 divides C proportionally among those ICN nodes (line 11). Then, the miss rate of each item in region r, $(\lambda'_{r,i})$, for all the data items and edge ICN nodes are calculated at line 12, depending on the replacement algorithm (i.e. cache models in Section 2.3 are used here). Next, the algorithm calculates the request rate at the parents of edge ICN node (lines 18 and 19). In the next step, algorithm divides C among edge ICN nodes as well as their immediate parents since the request rates at them all are known (line 20). The algorithm continues calculating miss rates and cache sizes for ICN nodes at higher levels incrementally in the ICN tree (line 17 to 25). The first iteration of algorithm, lines 8 to 26, results in: $\sum_{\forall r \in R} C_r > C$. So, the algorithm iterates the lines 8 to 26 until it gets to a stationary status in which $\sum_{\forall r \in R} C_r = C$.

5.4.2 Baseline Configuration

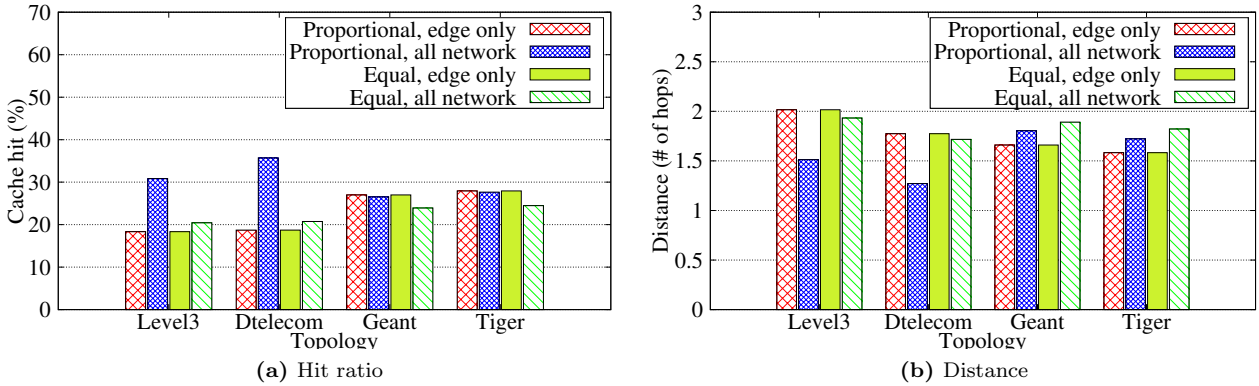
The behaviour of the system when the requests are identically distributed in each region is considered as the baseline configuration. Figure 5.8 shows the metrics for all four scenarios with LRU. For Dtelecom and Level3 topologies, scenario s_2 results in better performance while three metrics in Geant and Tiger topologies are similar in four different scenarios. As depicted in Figure 5.9, the same relative performance trends are seen with LRU and 2-LRU, although the latter algorithm outperforms LRU by about $16 - 37\%$ better in terms of hit ratio. 2-LRU also reduces average distance by $16 - 20\%$ and load on the content server by about $16 - 21\%$. The remainder of the experiments will only consider 2-LRU as it outperforms LRU.

5.4.3 Influence of Geographically Localized Traffic

Figure 5.10 depicts 2-LRU cache performance with geographic locality. Considering Dtelecom and Level3 topologies, cache budget distribution in s_1 and s_2 result in higher hit ratio, smaller distance and smaller load on the server. The reason for this similarity of cache performance between s_1 and s_2 is the high percentage of edge nodes in the two topologies (Table 3.1). The simulation results for Tiger and Geant show, however, that the system has slightly better performance in s_1 for all three metrics, as the error bars do not include the mean of the next best configuration (Scenario s_2). Scenarios s_3 and s_4 are worse, since they ignore the geographical locality. Their relative difference is also quite small for the server request rate, but larger for hit

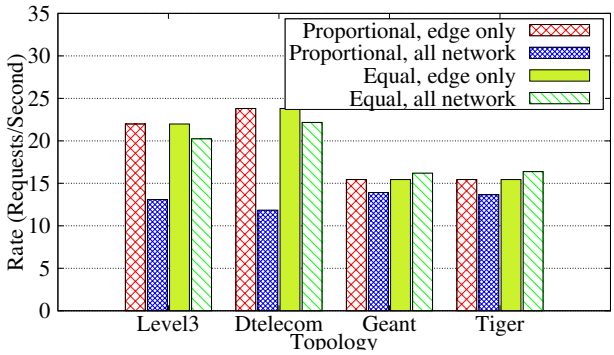
Algorithm 5.5 Dividing the cache budget proportionally among regions

procedure DIVIDE–CACHE–BUDGET
  INPUT:
    $C, R, R_r \quad \forall r \in R, \lambda_{r,i} \quad 1 \leq i \leq N \; \forall r \in R$
  OUTPUT:
    $C_r \quad \forall r \in R$
  initialize $\lambda_{r,i}$*:* $\lambda_{r,i} \leftarrow 0 \quad \forall r \in R - E$
  initialize C_r*:* $C_r \leftarrow 0 \quad \forall r \in R$
  while $\sum_{\forall r \in R} C_r \neq C$ **do**
    let A *be set of all edge nodes*
    $\lambda_r \leftarrow \sum_{i=1}^{N} \lambda_{r,i} \; \forall r \in R$
    $C_r \leftarrow \frac{\lambda_r}{\sum_{\forall t \in R} \lambda_t} C \quad \forall r \in A$
    calculate $\lambda'_{r,i}$, $\forall r \in A \; 1 \leq i \leq N$
    $B \leftarrow \{\}$
    find the parents of nodes in A*.*
    add those parents whose arrival rate at all their children nodes are known/calculated to B*,*
    such that $d \in A \rightarrow r \in B$, $\forall d \in D_r$
    while $A \neq R$ **do**
      $\lambda_{r,i} \leftarrow \sum_{\forall d \in D_r} \lambda'_{d,i} \; \forall r \in B \; 1 \leq i \leq N$
      $\lambda_r \leftarrow \sum_{i=1}^{N} \lambda_{r,i} \; \forall r \in B$
      $C_r \leftarrow \frac{\lambda_r}{\sum_{\forall t \in R} \lambda_t} C \; \forall r \in B$
      calculate $\lambda'_{r,i} \; \forall r \in B \; 1 \leq i \leq N$
      $A \leftarrow A \cup B$
      $B \leftarrow \{\}$
      add r *to* B *such that* $d \in A \rightarrow r \in B$, $\forall d \in D_r$
    end while
  end while
end procedure



(a) Hit ratio

(b) Distance



(c) Request rate at the server

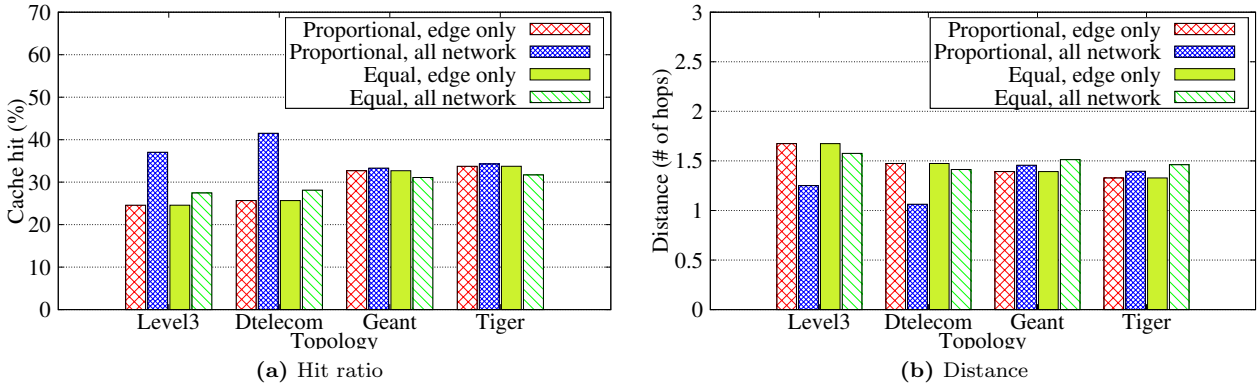
Figure 5.8: Identical Zipf distribution for all regions, LRU

ratio and distance. The findings of Figure 5.10 is in contrast with the findings in 5.9 in which cache budget distribution in s_2 results in better caching performance. Figure 5.10 also shows a significant improvement from Figure 5.9 in all metrics, because the differential α and rate of regional requests can leverage the differential cache allocation. This basic proof-of-concept shows that a more realistic localized model can influence cache performance metrics.

Figure 5.11 compares the average hit ratio at edge and intermediate ICN nodes for different topologies in scenario s_2. With geographical locality, intermediate ICN nodes obtain only a $5-10\%$ hit ratio. This is ineffective, consistent with Figure 5.10. For Dtelecom and Level3 topologies, with identical distributions for all users, the hit ratio at intermediate ICN nodes jumps to 37% and 26% respectively. This makes the overall hit ratio in s_2 higher than overall hit ratio in s_1 as illustrated before in Figure 5.9 that is inconsistent with the findings in Figure 5.10. One reason for the inefficiency of caching at intermediate ICN nodes is that arrival requests at intermediate ICN nodes, (the aggregate of edge misses), does not follow a Zipf distribution. This makes both candidate cache replacement algorithms inefficient.

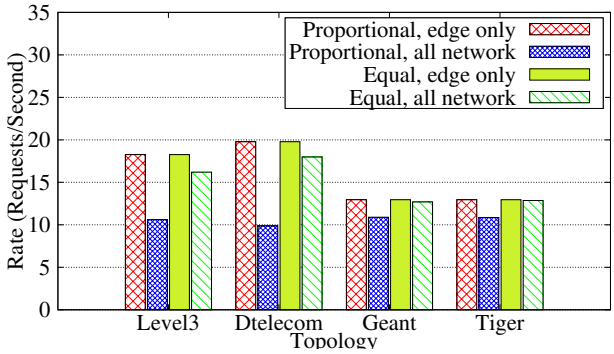
The findings of this section are summarized as following:

- Studying the performance of caching in ICNs with unrealistic synthetic distribution for users' requests results in misleading observations (Figure 5.10 vs Figure 5.9).
- For request patterns with geographical locality, cache distribution policy in s_2 results in a better caching



(a) Hit ratio

(b) Distance



(c) Request rate at the server

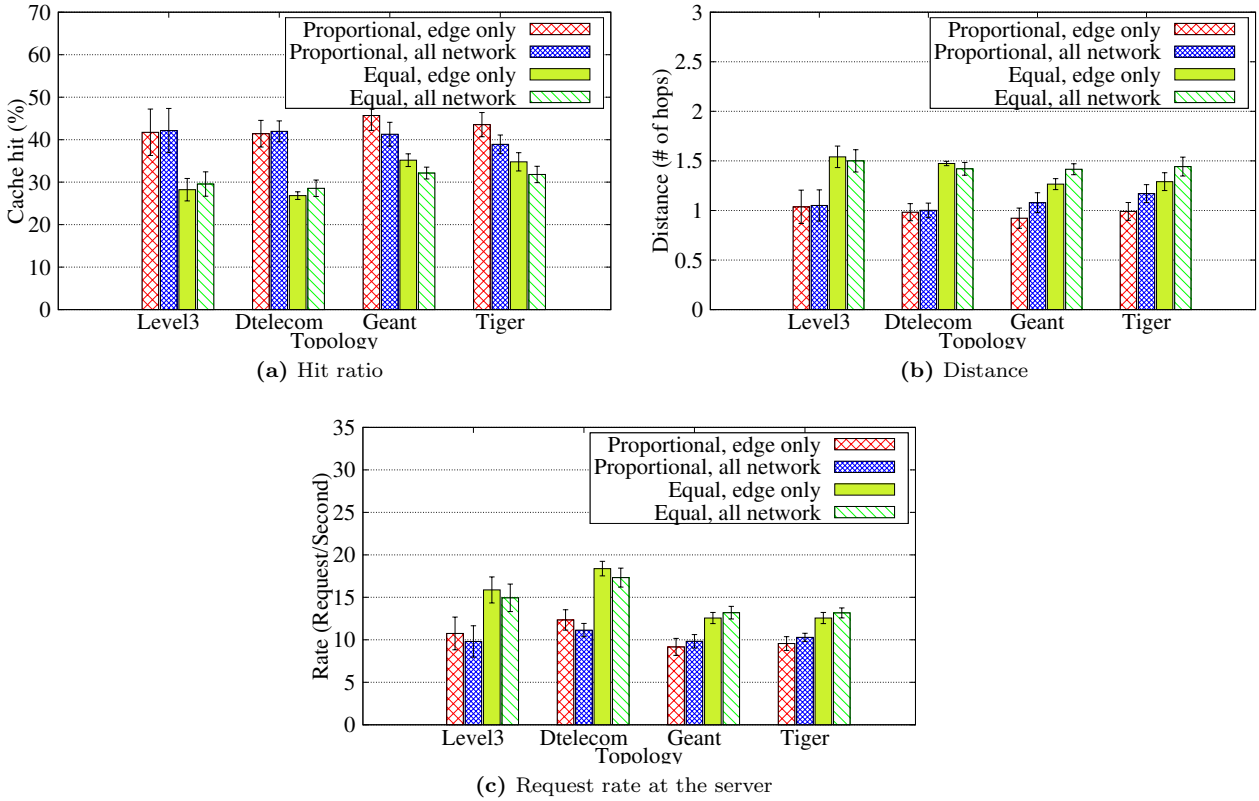
Figure 5.9: Identical Zipf distribution for all regions, 2-LRU

performance. This policy distributes the cache budget C among the edge nodes proportional to their arrival request rates. It is also found that caching at intermediate nodes results in very small hit ratio. This suggest that combination of other cache replication and replacement algorithms may result in better caching performance.

- Allocation of cache budget among the ICN nodes proportional to their arrival request rates may not be the optimal solution for maximizing the caching performance in ICNs. This will be studied in Chapter 6.

5.4.4 Data Lookup Assisted with Local Search

Figure 5.12 shows the effect of local search for $nsd = \{0, 1, 2\}$ for the four topologies with 2-LRU when users' requests follow different geographical distributions. No local search occurs with $nsd = 0$. A larger value for nsd (larger neighbourhood) results in better performance for all three metrics. For $nsd = 2$, the hit ratio increases by 16%, 14%, 8% and 13% for Level3, DTelecom, Geant and Tiger topologies respectively. Having $nsd = 2$ also results in shorter distance as well as $21 - 40\%$ less load on the server, compared with no local search. Figure 5.12 indicates that a local search with small $nsd = 2$ among the neighbouring ICN nodes can help the users to access their data items of interest with a shorter delay. In addition, the load on server



(a) Hit ratio

(b) Distance

(c) Request rate at the server

Figure 5.10: Geographical locality, 2-LRU

decreases since the requests are fulfilled locally.

Issues with similar distribution for neighbouring regions

Algorithm 5.4 randomly divides a global popularity distribution of N data items into k distributions for k regions. The distributions for k regions have various Zipf parameter α, request arrivals λ and different order of data items sorted based on their popularity. Having Geant topology with $k = 10$ for example, Algorithm 5.4 divides a popularity distribution for 20000 data items with global $\lambda = 40$ and $\alpha = 1.0$ to 10 sub distributions. Table 5.2 shows the output of the algorithm for $\psi = 1.0$ and $\psi = 0.1$. The algorithm creates regions u_1, u_9 and u_{10} similar to each other for $\psi = 0.1$. The Zipf parameter α stays at 1.64 for region u_9 while α jumps from 0.68 to 1.68 for region u_{10} when ψ moves from 1.0 to 0.1. The Zipf parameter in u_1 also decreases from 1.01 to 0.79. The similarity between two regions is calculated using (5.10). Table 5.3 shows the effect of $\psi = 0.1$ on the similarity between regions u_1, u_9 and u_{10}. For example, $\psi = 0.1$ decreases the distance (increases the similarity) between regions u_9 and u_{10} by about 99%.

Table 5.4 depicts the influence of similarity on average hit ratio and retrieval distance for neighbouring

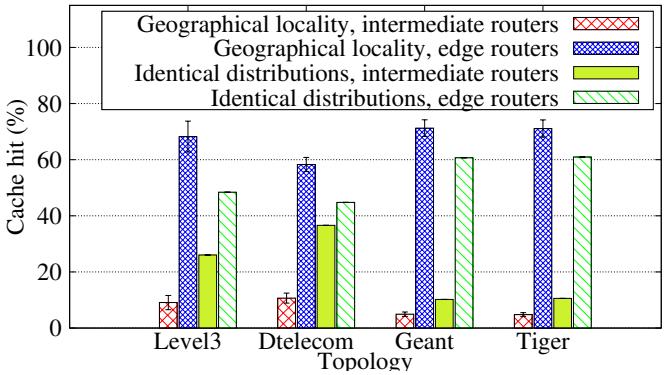
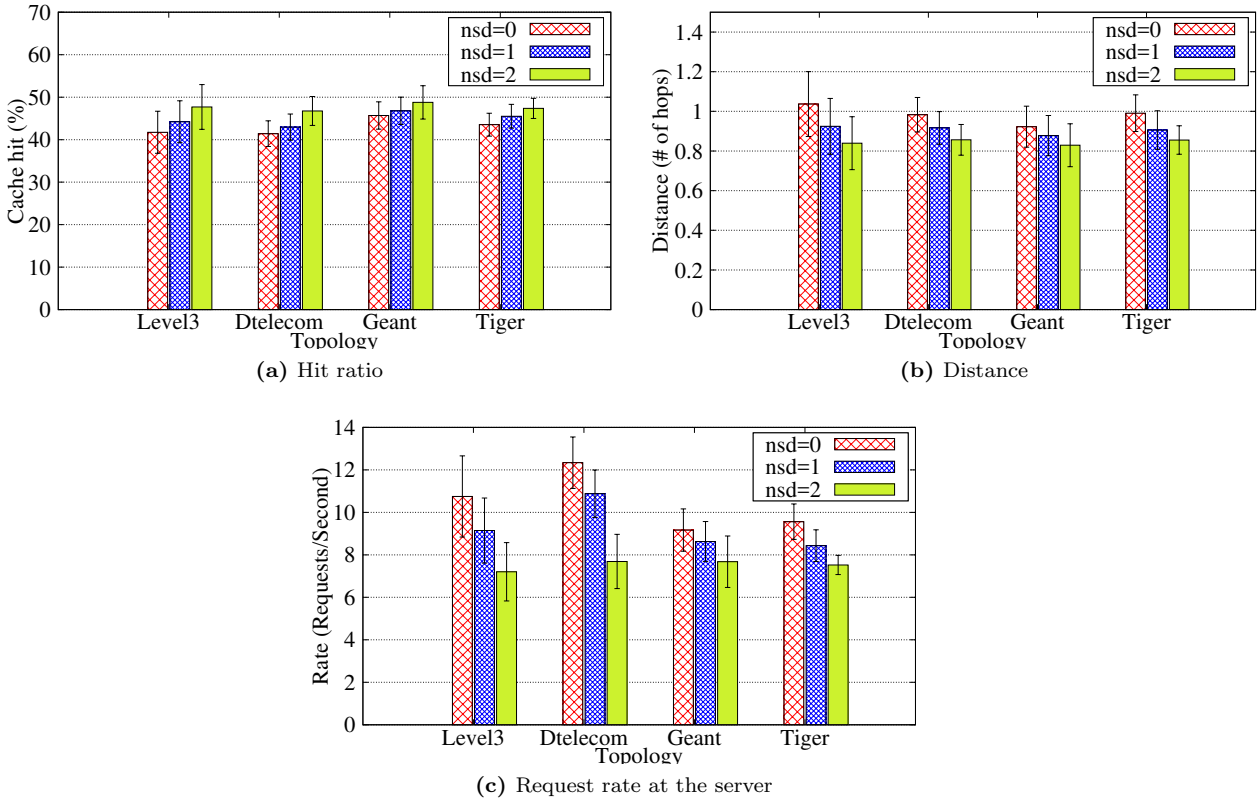


Figure 5.11: Edge ICN nodes vs intermediate ICN nodes in s_2, 2-LRU

Table 5.2: Output of Algorithm 5.4

u_x	$\psi = 1.0$		$\psi = 0.1$	
	λ_x	α_x	λ_x	α_x
u_1	36.55	1.01	30.38	0.79
u_2	0.01	0.71	0.01	0.71
u_3	0.01	1.87	0.01	1.87
u_4	0.01	1.96	0.01	1.96
u_5	0.01	1.87	0.01	1.87
u_6	0.39	1.55	0.39	1.55
u_7	0.01	1.51	0.01	1.51
u_8	0.23	1.76	0.23	1.76
u_9	0.78	1.64	6.14	1.64
u_{10}	2.02	0.68	2.83	1.68

regions u_9 and u_1 when $\psi = 0.1$ and $\psi = 1.0$. As illustrated in this table, in case of equal-edge-only (i.e. scenario for the distribution of cache budget among the ICN nodes) and $\psi = 1.0$, assisted local search for region u_9 with $nsd = 1$ in region u_1 decreases the retrieval distance by 19%. The retrieval distance decreases by 11%, however, when similarity between regions u_9 and u_1 increases. It can be implied that more similarity between the two regions decreases the performance of local search. The reason is that as the distributions of users' requests in two regions get closer, it means they have similar data items cached in their caches. As a result, a local search for data item i that is not in the u_9's cache may not be in the u_1's cache either. In case of using the proportional-edge-only scenario, a much larger cache is allocated to region u_9 when $\psi = 0.1$ compared to $\psi = 1.0$ (see Table 5.2); this increase in the u_9's cache budget decreases the retrieval distance from 0.48 to 0.12 in case of no local search. Despite this change, the relative benefit of $nsd = 1$ over $nsd = 0$ in regards with the retrieval distance is about 42% for both $\psi = 1.0$ and $\psi = 0.1$.



(a) Hit ratio

(b) Distance

(c) Request rate at the server

Figure 5.12: Geographical with local search, 2-LRU

For region u_1 on the other hand, different results are observed. It seems that local search helps more when similarity between regions u_1 and u_9 increases. This behaviour of u_9's cache however, is due to the change in Zipf parameter α for the users' requests in u when ψ changes from 1.0 to 0.1. Smaller α means more data items with similar popularity at the head of Zipf distribution that results in more misses. Therefore, for a smaller α for the Zipf distribution of users' requests in region u_1, the cache hit ratio drops from 0.70 to 0.43 in equal-edge-only scenario and from 0.69 to 0.40 in proportional-edge-only scenario. This smaller α in region u_1 when $\psi = 0.1$ causes a high competition between data items with similar frequency to stay in u_1's cache. As a result, a miss for popular data item i at u_1's cache will be probably a hit at u_9's cache since i is a popular data item in u_9 region as well that makes the local search a little more effective compared to $\psi = 1.0$ where fewer data item need to compete to stay in cache (larger α).

These results show that studying the effect of similar distributions for neighbouring regions on the overall caching performance of a network of caches is complicated since the shape of distributions (Zipf parameter α) does not stay constant when parameter ψ changes in Algorithm 5.3. The reason is that Algorithm 5.1 cannot find a Zipf distribution for region u_1 for example with the same α when creating region u_1 similar to region u_9. Modifying Algorithm 5.1 so that it creates similar regions through changing only the rank of data items while Zipf properties of the regions left unchanged could be a part of future work. In this case, studying the performance of similar distributions for users' requests on local search for neighbouring regions

Table 5.3: Distance between regions u_1, u_9 and u_{10}.

ψ	u_9 *vs* u_1	u_{10} *vs* u_1	u_{10} *vs* u_9
1.0	0.238	0.228	0.345
0.1	0.002	0.003	0.004

Table 5.4: Influence of ψ on local search; (e for *lsd* shows means the percentage of decreased retrieval distance for *lsd*=1 compare to *lsd*=0)

		$\psi = 1.0$				$\psi = 0.1$			
		distance			hit ratio	distance			hit ratio
	cache allocation	$nsd = 0$	$nsd = 1$	relative benefit	1=2	$nsd = 1$	$nsd = 1$	relative benefit	1=2
u_9	Equal-edge-only	0.06	0.05	19%	0.99	0.04	0.03	11%	0.99
	Proportional-edge-only	0.48	0.28	41%	0.88	0.12	0.07	42%	0.97
u_1	Equal-edge-only	1.21	1.17	3%	0.70	2.28	2.10	7%	0.43
	Proportional-edge-only	1.24	1.24	0%	0.69	2.40	2.39	0.3%	0.40

is possible.

5.5 Summary

In this chapter, an algorithm is proposed to generate locally biased request patterns which follow different Zipf distributions for each region, and combine to form a Zipf distribution. Then, the performance of the system is studied in four different scenarios. Simulation results show that geographical locality causes the different system behaviour in the simple test scenarios compared to identical request distribution. A local search is also introduced to ICN networks. Using a local search alongside the standard content discovery/delivery mechanism provides higher hit ratios, shorter distances and lighter system load on the server. A very important finding in this chapter is that in-network caching has little advantage compared to caching only in edge ICN nodes. This will be studied in more details in the next chapter.

The proposed traffic generator shows different caching behaviour in existence of geographical locality in users' requests. In addition to geographical locality, neighbouring regions can also have a similar users' requests. The similarity of request patterns in neighbouring regions could affect the caching performance in network of caches. A limitation to study this influence here is that there is no quantitative amounts by which the regions differ and how much two cities in a state will be more similar than two cities in different countries.

Note that portions of this chapter have been published at *LCN 2017* [65], and a full conference paper is in preparation.

Chapter 6

Optimal Cache Budget Distribution For Hierarchical Trees of ICN Nodes

6.1 Motivation

Caching facilities in in-network caching can be deployed by all (e.g. CCN [49]) or some (e.g. centrality-based caching[18]) of the ICN nodes on the path of delivering data items from the source to the users. Some studies in the past few years have investigated the efficiency of caching in ICN, both experimentally, using logs at CDNs [37] and content publishers [91], and analytically [66, 71], from which some have inconsistent results. For example, Danzig *et al.* [32] and Rossini *et al.* [79] have shown that in-network caching can be more effective. Fayazbakhsh *et al.* [37] and Psaras *et al.* [72] on the other hand believe caching closer to the network edge is more effective.

This suggests that ICN literature still lacks an empirical and analytical deep understanding of benefits brought by in-network caching; e.g. Fayazbakhsh *et al.* [37], Dabirmoghaddam *et al.* study the in-network of caches under IRM [28]. Jia *et al.* introduce a model using two dimensional discrete Markov chain to calculate the hit ratio of ICN nodes in a two-level topology [50]. Some studies introduce a linear optimization problem to find the optimal ICN nodes in the network on which data item i needs to be cached in order to get a maximum benefit out of the network [13, 54, 58, 95, 70, 29].

Assuming R ICN nodes in the network, out of which E are connected to the users, C_j as the cache capacity of ICN node j, and N as the number of data items, the optimal caching in such works is modelled as

$$\text{Minimize} \sum_{l=1}^{E} \sum_{j=1}^{R} \sum_{i=1}^{N} s_i p_{li} c_{lj} x_{lji}, \text{[95]}$$

$$\text{subject to:} \begin{cases} \sum_{i=1}^{N} s_i x_{li} \leq C_l \quad l = 0, \ldots, E, \\ x_{lji} \leq x_{ji} \quad l = 1, \ldots, M, l \neq j = 0, \ldots, E, \quad \forall i, \\ \sum_{j=1}^{R} x_{lji} \geq 1 \quad \forall i, \forall l, \end{cases}$$

in which, c_{lj} shows the cost of retrieving a data item at ICN node j from ICN node l, p_{li} shows the popularity ratio for item i at ICN node l, s_i depicts the size of item i, and x_{ljl} indicates whether requests for data item i at ICN node l are served from the cache of ICN node j or not. Variable x_{li} also indicates whether data item i is stored in the cache of an ICN node l or not. The first constraint means the total size of cached items at ICN node l's cache should not be larger than C_l. The second constraint depicts that data item i can be retrieved from ICN node j if a copy of data item i is cached at ICN node j. Finally the last constraint means at least one copy of data item i should be cached somewhere in the network. The solution for this optimization problem are x_{ji}, $1 \leq i \leq N$, $1 \leq j \leq R$ indicating whether data item i should be cached at ICN node j in order to minimize the overall cost of retrieving data items. This model however ignores the effect of cache replacement and replication algorithms on the caching performance of the ICNs as well as their analytical findings are all under IRM.

Rizk *et al.* study the cache budget distribution in a two-level tree under IRM [74]. They use Garetto's model (2.21) and (2.22) to estimate the hit ratio in such a tree where LRU and LCE are used as cache replacement and replication algorithms. They find out that allocating a larger fraction of the cache budget to the root of this two-level tree increases the hit ratio in the system. They also find that allocating all the cache budget to the edge nodes does not minimize the average distance to retrieve data items. Dabirmoghaddam *et al.* find the optimal cache budget distribution among the levels of k-ary trees where LRU and LCE are used as cache replacement and replication algorithms [28]. Their optimal cache distribution minimizes the expected time to access data items that is a metric from users' point of view. Their study shows that in-network caching brings no benefit in shortening the distance from users to the first copy of data items.

To investigate the benefits of in-network caching, an analytical model is proposed in this chapter. The model optimally distributes a total cache budget of C among the nodes of ICN networks for LRU cache replacement algorithm and LCE replication algorithm. The proposed Algorithm 5.4 in Chapter 5 and Garetto's model [45] (explained in detail in Section 2.4) are used to apply the geographical and temporal localities, respectively, in users' requests.

The rest of this chapter is organized as follows: Section 6.2 proposes the evaluation model to analyze the cache performance in hierarchical trees of ICN nodes; a mathematical expression for the optimal cache distribution is also given in this section. Section 6.3 investigates the optimal cache distribution among the nodes of an ICN for LRU and LCE cache replacement and replication algorithms respectively for various topologies and network metrics. Finally, Section 6.4 summarizes this chapter.

6.2 Model and Assumptions

In the rest of this chapter, the following assumptions are taken into account:

1. The original copies of all data items are hosted by one source node.

2. ICN constructs an overlay tree consisted of all ICN nodes, rooted at the source node, for the content

discovery/delivery mechanism (explained in Chapter 1).

3. Users are only connected to the edge ICN nodes. In other words, edge ICN nodes receive exogenous traffic, while intermediate ICN nodes receive only endogenous traffic.

4. Garetto's SNM model [45] is used to apply temporal locality.

5. Users' requests have geographical locality. Algorithm 5.4 is used for this purpose.

6. LRU is used as the cache replacement algorithm.

7. LCE is used as the cache replication algorithm.

The optimization problem (6.9) that will be introduced is a non-linear problem that is difficult to solve. It is intended to keep it simple as much as possible. In this regard, considering other cache replication and replacement algorithms with more complicated models (e.g. 2-LRU's model (2.9) and LCD's model (2.19), (2.20)) results in a complicated model that could not be solved. The optimization problem (6.9) that is based on LRU cache replacement and LCE cache replication algorithms has one local solution that is also the global solution. Therefore, a mathematical software like Matlab can solve this optimization problem to find that global optimal solution. For combinations of other cache replacement and replication algorithms, (e.g. LRU and LCD, 2-LRU and LCE and 2-LRU and LCD) the corresponding optimization problems have several local solutions. Therefore, Matlab fails to find the optimal solution. As a result of the complexity of optimization problem for other replacement and replication algorithms, this chapter is focused on LRU and LCE that are used by other researchers [28, 74].

First, the evaluation model to analyze the cache performance in a hierarchical tree of ICN nodes is proposed. The hit probability of item i at cache k, depicted by $P_{k,i}$, is a function of $\lambda_{k,i}$ and C_k; i.e. $P_{k,i} := f(\lambda_{k,i}, C_k)$. Then, the matrix model of request propagation of data item i is presented here. Having the notation in Table 3.2, the propagation of each request for data item i in a hierarchical tree of caches is expressed by the following model:

$$\Lambda_i^{t+1} = M_i^t.\Lambda_i^t + O_i. \tag{6.1}$$

Λ_i is an $|R| \times 1$ matrix consisting of the request rates for item i (i.e. $\Lambda_i := \left[\lambda_{k,i}\right]_{|R|\times 1}$). O_i is the $|R| \times 1$ matrix whose elements are the exogenous request rates of data item i at ICN nodes as

$$\left[O_i\right]_k := \begin{cases} \lambda_{k,i} & \forall k \in E \\ 0 & otherwise, \end{cases}$$

and M_i is the $|R| \times |R|$ matrix of request propagation for data item i as

$$\left[M_i\right]_{kl} := \begin{cases} 1 - f(\lambda_{l,i}, C_l) & \forall l \in R_k \\ 0 & otherwise. \end{cases} \tag{6.2}$$

Having $\Lambda_i^0 = O_i$, for a hierarchical tree of caches with depth of d, it is required to perform the calculation of Λ_i^t for $t = \{1, 2, \ldots, d\}$; then, Λ_i^d has the request rates for item i at all regions of the nodes in the hierarchical tree. The goal is finding the optimal distribution of the cache budget among the caches of the network considering the following metrics:

- Distance, a metric from users' point of view: the data items should be cached as close as possible to the edge.
- Miss ratio, a metric from ISP's point of view: the overall miss ratio in the network should be minimized. A less overall miss ratio in an ICN network imposes a smaller load on the server.

Having these two metrics, the optimization problem is stated as

$$\underset{C_1,\ldots,C_{|R|}}{\text{minimze}} \begin{cases} \sum_{k=1}^{|E|} d_k(C_k) \\ \frac{\sum_{k=1}^{|R|} \lambda'_k(C_k)}{\sum_{k=1}^{|R|} \lambda_k} \end{cases} \tag{6.3}$$

$$\text{subject to:} \begin{cases} \sum_{k=1}^{|R|} C_k = C \\ C_k \geq 0 \quad \forall k, \end{cases} \tag{6.4}$$

in which $\lambda'_k(C_k)$ is the overall miss rate at region k, given by

$$\lambda'_k(C_k) = \sum_{i=1}^{N_k} \lambda_{k,i}\big(1 - f(\lambda_{k,i}, C_k)\big) \quad \forall k, \tag{6.5}$$

and $d_k(C_k)$ is the average distance for retrieving data items from node k, given by

$$d_k(C_k) = \frac{1}{\sum_{i=1}^{|E|} \lambda_k} \sum_{i=1}^{N_k} \lambda_{k,i} d_{k,i}(C_k) \quad \forall k, \tag{6.6}$$

where $d_{k,i}$ as the average distance to retrieve data i from edge node k is calculated as

$$\begin{aligned} d_{k,i}(C_k) = \big(1 - f(\lambda_{k,i}, C_k)\big) f(\lambda_{k^1,i}, C_{k^1}) + 2\big(1 - f(\lambda_{k,i}, C_k)\big)\big(1 - f(\lambda_{k^1,i}, C_{k^1})\big) f(\lambda_{k^2,i}, C_{k^2}) \\ + \cdots + D_k\big(1 - f(\lambda_{k,i}, C_k)\big)\big(1 - f(\lambda_{k^1,i}, C_{k^1})\big)\big(1 - f(\lambda_{k^2,i}, C_{k^2})\big) \ldots \big(1 - f(\lambda_{k^{D_k-1},i}, C_{k^{D_k-1}})\big). \end{aligned} \tag{6.7}$$

Equation (6.7) is simplified to

$$d_{k,i}(C_k) = \sum_{l=0}^{k^l \neq s} \prod_{j=0}^{l} \big(1 - f(\lambda_{k^j,i}, C_{k^j})\big). \tag{6.8}$$

Now, the two objective functions are combined as follows:

$$\underset{C_1,\ldots,C_{|R|}}{\text{minimze}} \left\{ (1-w) \frac{\sum_{k=1}^{|E|} d_k(C_k)}{D} + w \frac{\sum_{k=1}^{|R|} \lambda'_k(C_k)}{\sum_{k=1}^{|R|} \lambda_k} \right\}, \tag{6.9}$$

in which D normalizes the first objective and w is the weighting coefficient.

6.3 Numeric Results

To study the benefits of caching in the network, named Network Overall Caching (NOC), over Edge-Only Caching (EOC), in terms of overall miss ratio and retrieval distance, the two following metrics are considered:

- Relative benefit of NOC over EOC in terms of load on the server: assuming h_{EOC} and h_{NOC} as the average load on the server in EOC and NOC respectively, this metric would be

$$\frac{h_{EOC}-h_{NOC}}{h_{EOC}} .$$

- Relative benefit of NOC over EOC in terms of retrieval distance: assuming d_{EOC} and d_{NOC} as the average retrieval distance in EOC and NOC respectively, this metric would be

$$\frac{d_{EOC}-d_{NOC}}{h_{EOC}} .$$

The optimal cache budget distribution is solved for the following three scenarios:

- geographical locality and strong temporal locality: for this scenario, Algorithm 5.4 is used to implement geographical locality. To apply the temporal locality, $z = 10$ is chosen for the second order hyper-exponential process (2.14).

- geographical locality and weak temporal locality: for this scenario, Algorithm 5.4 is used to implement geographical locality. To apply the temporal locality, $z = 2$ is chosen for the second order hyper-exponential process (2.14).

- geographical locality and no temporal locality: for this scenario, Algorithm 5.4 is used to implement geographical locality. Having $z = 1$ for the second order hyper-exponential process (2.14) implies no temporal locality.

The optimization problem (6.9) is solved in Matlab for $N = 10000$, the global request rate $\lambda_r = 4$, and various values of C, α and topologies. The parameters for Algorithm 5.4 are the same used in Section 5.4.1. Since Algorithm 5.4 generates different outputs for each run, 5 different outputs of this algorithm is used to solve the optimization problem for each topology.

6.3.1 Tree Topologies

In this section, the optimal cache budget distribution for a ternary tree with depth of 4 and different values of Zipf parameter α and C is solved; $l = 1$ shows the root of the tree and largest value of l corresponds to the level of edge nodes. Figure 6.1 shows the optimal cache distribution among different levels on this 4-level ternary tree for different values of α. For $\alpha = 0.8$ (Figure 6.1a), as w moves from 0 to 1, meaning more weight is given to minimizing the overall miss ratio of the objective function (6.9), a large fraction of cache budget

is distributed among high level intermediate nodes. For example, 45% of the cache budget is allocated to the root of the tree for $z = 1$ and $w = 0$. The fraction of cache budget allocated to the root however, increases to 100% for $z = 1$ and $w = 1$.

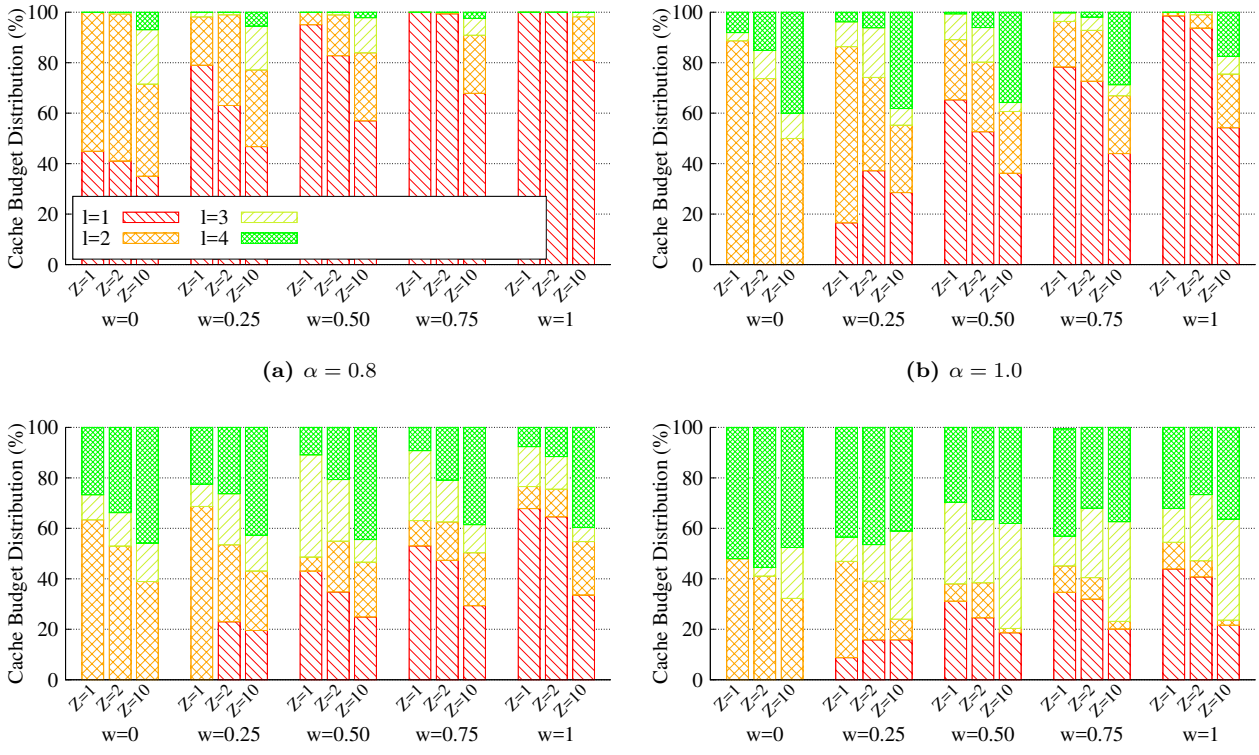
The other point from Figure 6.1 is that, as stronger temporal locality is applied to the user's request ($z = 10$), allocated cache budget among lower level caches increases. In case of $\alpha = 0.8$ for instance, 95% and 5% of the cache budget is allocated to first level and second levels of the tree respectively for $w = 0.5$ and $z = 1$. The cache distribution for the levels changes to 57%, 27%, 14% and 2% for $w = 0.5$ and $z = 10$. This means that for the scenario with strong temporal locality, caching items at the lower level ICN nodes results in smaller retrieval distance as well as smaller load on the server. This trend for w and z is also seen for other values of α (Figure 6.1b, 6.1c and 6.1d).

Comparing Figures 6.1a, 6.1b, 6.1c and 6.1d against each other illustrates the effect of α on optimal distribution of the cache budget in the tree. As α increases, a larger fraction of the cache budget is scattered among lower level caches in order to minimize the objective function (6.9). For instance, when $w = 0.50$ and $z = 1$, no fraction of the cache budget is allocated to the two lowest levels of the tree for $\alpha = 0.8$. The third and fourth levels however, consume more than 60% of the cache budget together for $\alpha = 1.4$ to minimize (6.9).

Figures 6.2 and 6.3 represent the relative benefit of optimal in-network caching over edge caching for average distance and the load on the server respectively. Figure 6.2a shows that the optimal in-network caching for $z = 1$ and $w = 0$ decreases the average retrieval distance by 5.5%, 8.3%, 9% and 10% when α equals to 0.8, 1, 1.2 and 1.4 respectively. This benefit rises as temporal locality gets stronger; for instance for $z = 10$ and $w = 0$, the optimal in-network caching decreases the average retrieval distance by 11.4%, 11.3%, 13.7% and 18.3% when α equals to 0.8, 1.0, 1.2 and 1.4 respectively. Figure 6.2 also shows that the benefit of in-network caching on retrieval distance goes down as w increases. For $w = 1.0$ for instance, a high fraction of the cache budget is allocated to the ICN nodes at higher level; this increases the average retrieval distance. However, the performance of in-network caching does not get worse than edge-only caching as the relative benefit of in-network caching over edge-only caching stays positive for all values of z and α.

On the other hand, Figure 6.3 depicts the influence of optimal caching on load on the server. The lower miss ratio, the lighter load on the server. The figure shows that the optimal in-network caching in the tree under study provides at least 23.2% less load on the server ($z = 1$, $\alpha = 0.8$ and $w = 0$). This benefit goes up close to 80% for strong temporal locality of $z = 10$ and $\alpha = 0.8$, $w = 1.0$.

Having $\alpha = 1.0$ and considering an equal weight for minimizing the average retrieval distance and minimizing the overall miss ratio of the network, $w = 0.5$, the optimal in-network caching decreases the retrieval distance by 6.4%, 6.9% and 9.9% for $z = 1$, $z = 2$ and $z = 10$ respectively. The optimal in-network caching however, decreases the average load on the server by 43%, 42% and 62.9% for $z = 1$, $z = 2$ and $z = 10$ respectively. These results show that while in-network caching may have a small influence on the average retrieval distance (up to 9.9%), network caching can be very effective on minimizing the load on the server



(a) $\alpha = 0.8$ **(b)** $\alpha = 1.0$

(c) $\alpha = 1.2$ **(d)** $\alpha = 1.4$

Figure 6.1: Optimal cache budget distribution, $C = 1000$

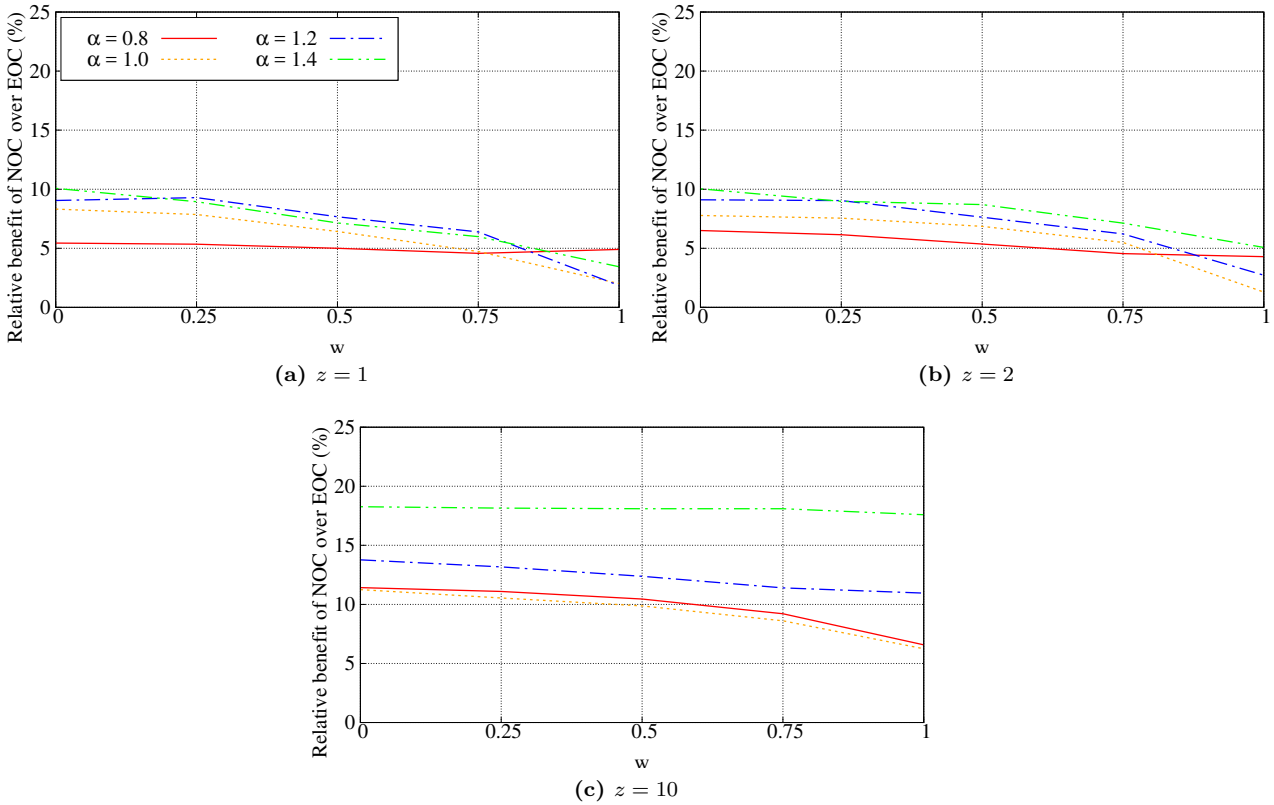
(at least 42%).

Figure 6.4 represents the influence of C on optimal in-network caching. Similar to Figure 6.1, a larger value of w allocates a larger proportion of C to caches at the intermediate levels; in addition, stronger temporal locality requires more cache budget at lower levels of tree to minimize the objective function (6.9). Comparing Figures in 6.4 indicates that optimal in-network caching distributes a larger fraction of C among lower levels as C increases. For instance for $z = 2$ and $w = 1$, more than 90% of cache budget is allocated to the first level of the tree when $w = 1000$. The fraction of cache budget allocated to the first level decreases to less than 70% when $C = 5000$.

Fiugre 6.5 shows the benefits of optimal in-network caching on average retrieval distance for different values of z. This figure shows that strong temporal locality causes a large difference in gained relative benefit of in-network caches for different cache budgets. Having $w = 0.5$ and $z = 10$ for example, the benefit of in-network caching on retrieval distance when $C = 5000$ is 22% compared to benefit of in-network caching when $C = 1000$ which is 10%. This difference shrinks to negligible values when $z = 1$. A similar behaviour is also seen for the benefit of in-network caching on overall load on the server in Figure 6.6.

Findings

- Assuming $C = 1000$ and $\alpha = 1.0$:



(a) $z = 1$ **(b)** $z = 2$

(c) $z = 10$

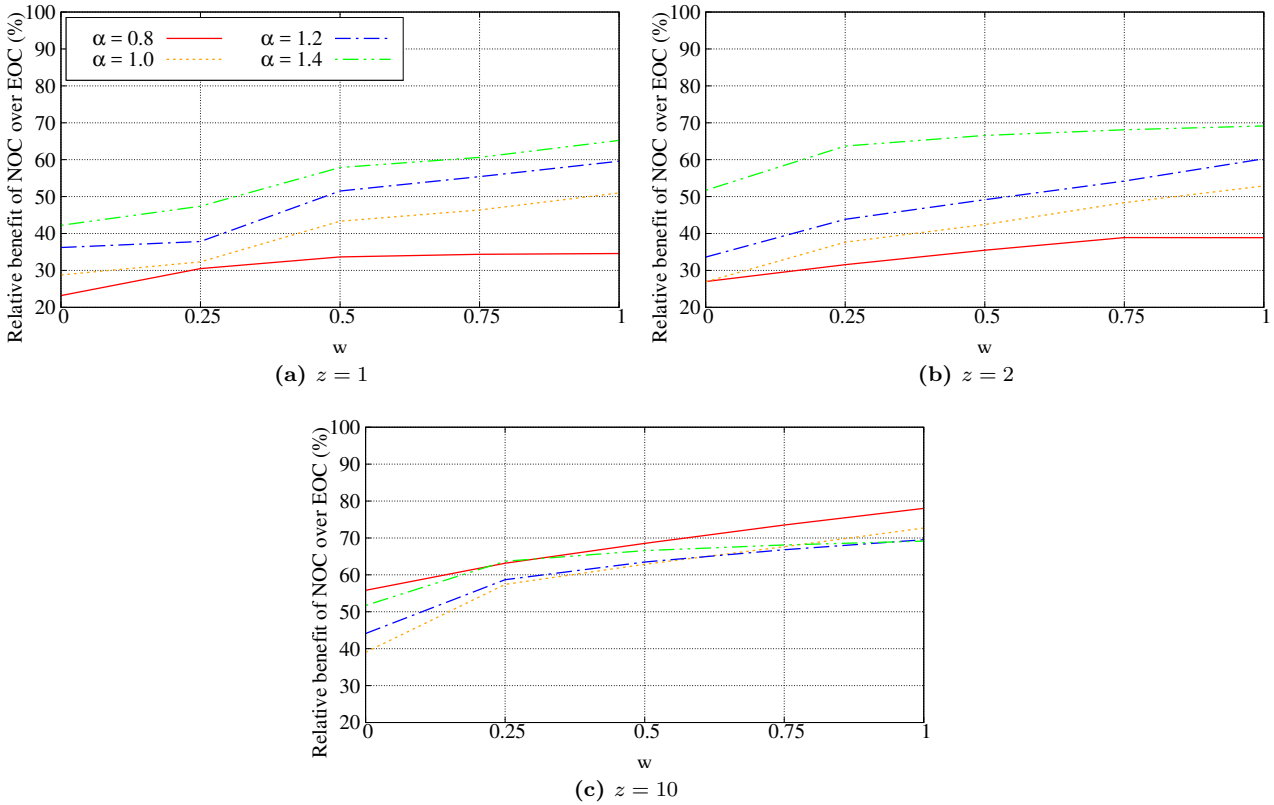
Figure 6.2: Relative benefit of NOC over EOC, average distance, $C = 1000$

- From the ISP's point of view, caching data items closer to the source results in lower overall miss ratio. This results in less traffic load on the server. Optimal in-network caching can decrease the load on the server by at least 50% (when $w = 1$ and $z = 1$).
- From users' point of view, optimal in-networking caching results in up to 12% shorter retrieval distance compared to edge-only caching for $z = \{1, 2, 10\}$.

- For data items with stronger temporal locality, optimization problem (6.9) distributes a larger fraction of C among levels closer to the edge.
- For request distributions with larger values of α, optimization problem (6.9) distributes a larger fraction of C among levels closer to the edge.
- For smaller cache budgets, optimization problem (6.9) distributes a larger fraction of C among levels closer to top levels of the tree.

6.3.2 Realistic Topologies

In this section, the optimization problem (6.9) is studied for realistic topologies in Table 3.1. Figure 6.7a shows that optimal in-network caching in Geant topology may not allocate any fraction of C to some intermediate nodes; that is consistent with the findings of Chai's study [18]. Minimizing the retrieval distance for users



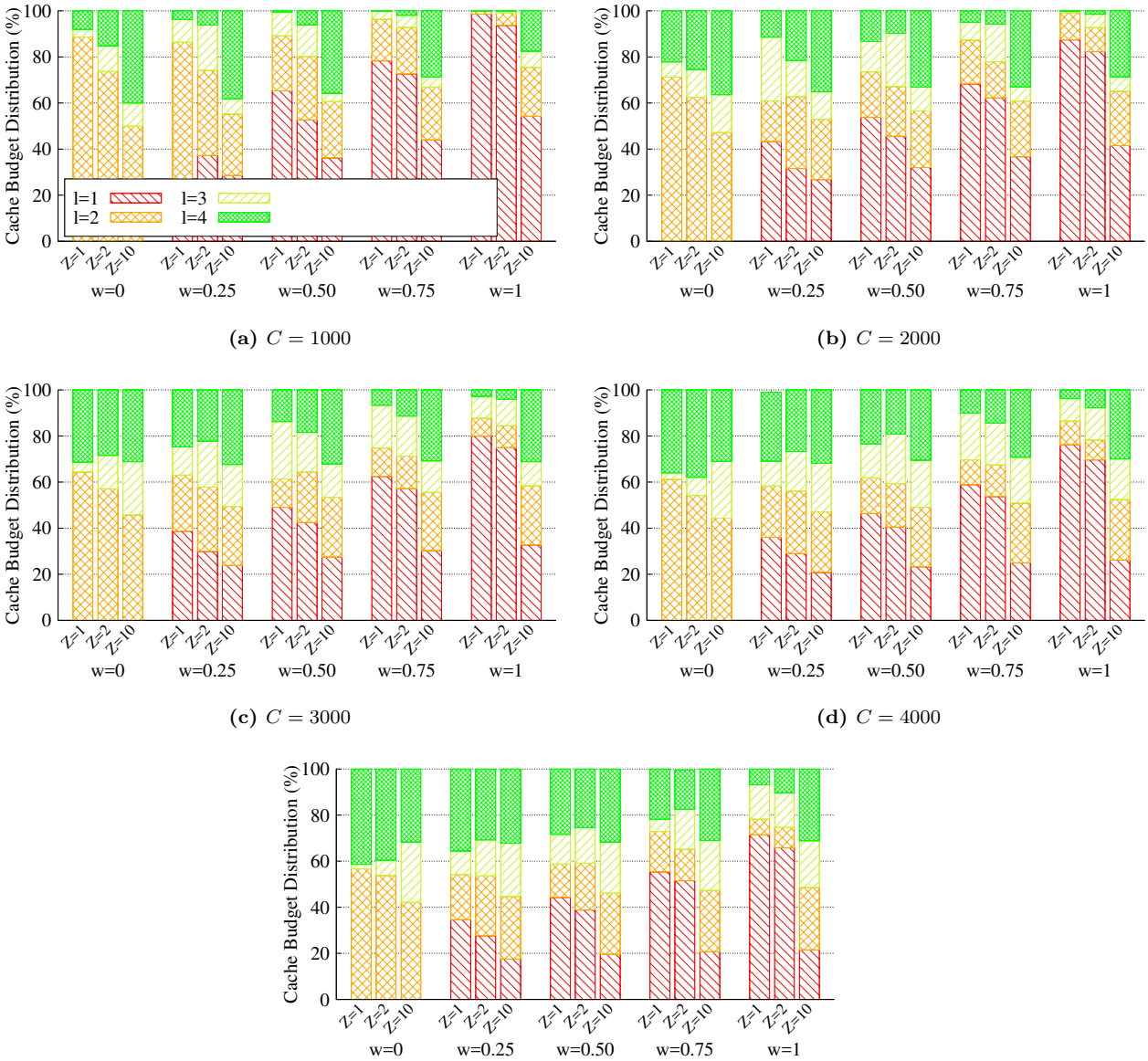
(a) $z = 1$ **(b)** $z = 2$

(c) $z = 10$

Figure 6.3: Relative benefit of NOC over EOC, average load on server, $C = 1000$

($w = 0$) for example, allocates more/less than 60%/40% of C to the edge nodes and nodes at the second level of the topology respectively; no cache would be allocated to any other levels of Geant topology. Similar to the findings of the previous section, a larger fraction of C would be allocated to higher levels as w gets closer to 1; however, no cache budget is yet distributed to levels 3, 4 and 5 for different strengths of temporal locality when $w = 1$. This means that caching at some levels of Geant topology brings no benefit to minimize the objective function (6.9). Figure 6.7b depicts the influence of w on superiority of in-network caching over edge-only caching for average retrieval distance. The maximum benefit brought by in-network caching is less than 10% when $w = 0$. In-network caching fails to be effective on minimizing the retrieval distance as w gets greater than 0.75. Figure 6.7c on the other hand illustrates the influence of w on superiority of in-network caching over edge-only caching for overall load on the server. This figure shows that the in-network caching provides the network with at least 15% less load on the server when $w = 0$. This goes up to at least 40% when $w = 1$. Choosing w here depends on how much smaller retrieval distance could be perceived by users. For instance, if less than 10% shorter retrieval distance has no influence of users' experience, $w = 0.75$ can be selected to guarantee at least 30% less load on the server.

Figure 6.8 depicts the optimal distribution for Tiger topology. Like Geant topology, caching in some levels of Tiger topology, e.g. levels 2 and 3, has no effect on the optimization problem (6.9). Having $w = 0.5$ results in at least 30% less load on the server while having less than 5% influence on the users' experience.



(a) $C = 1000$

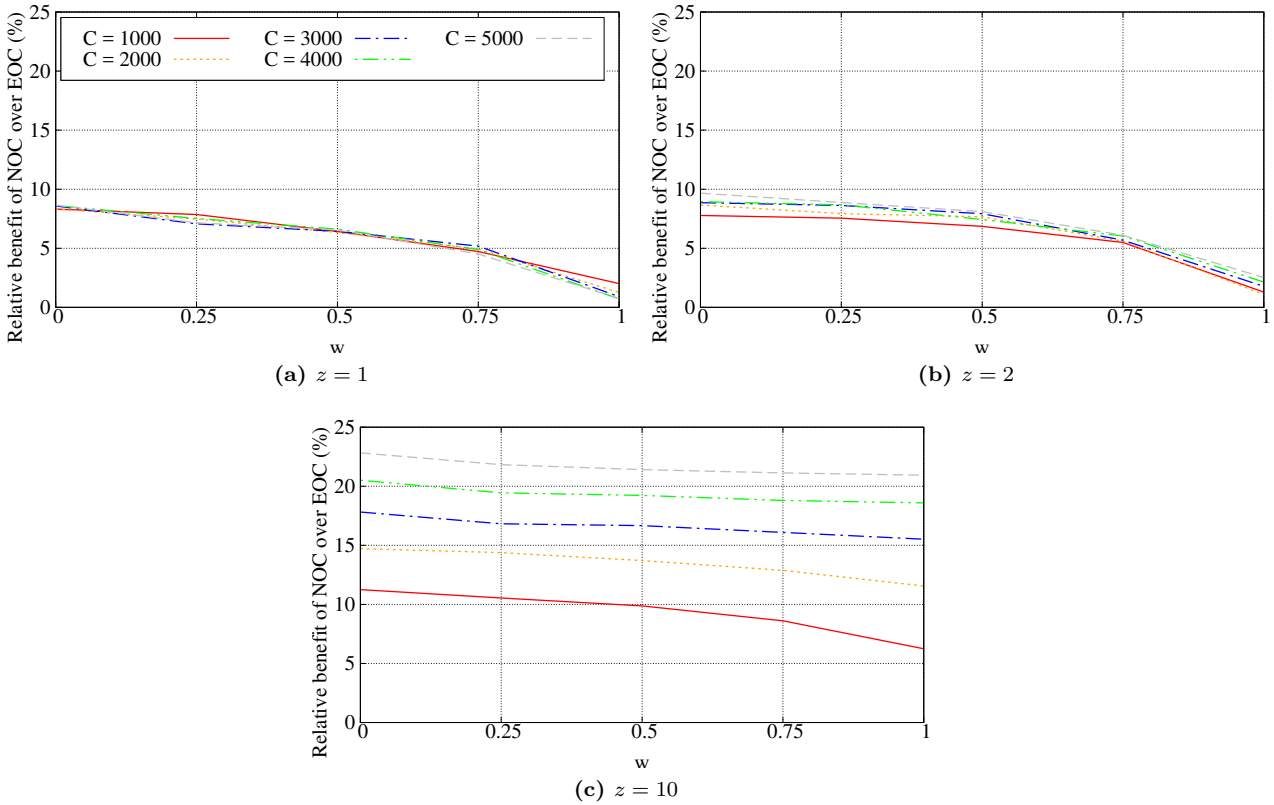
(b) $C = 2000$

(c) $C = 3000$

(d) $C = 4000$

(e) $C = 5000$

Figure 6.4: Optimal cache budget distribution, $\alpha = 1.0$



(a) $z = 1$ **(b)** $z = 2$

(c) $z = 10$

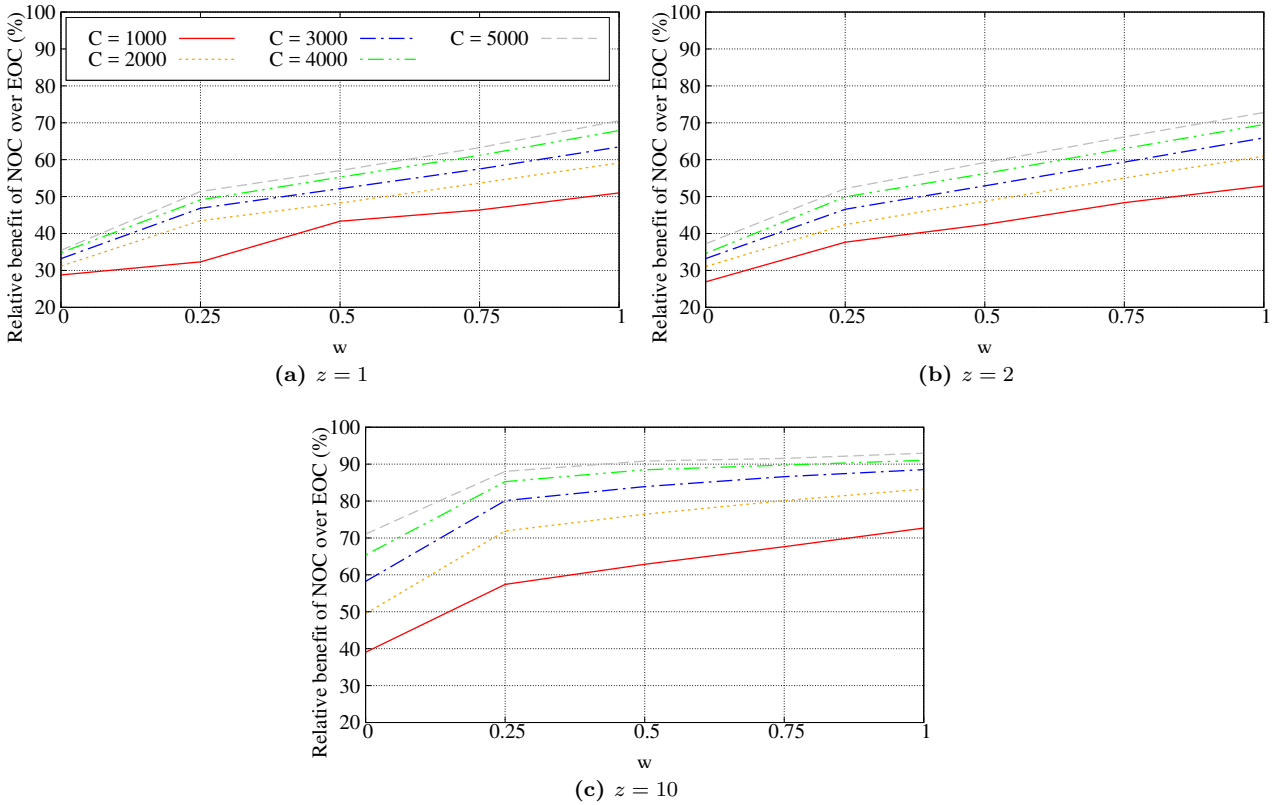
Figure 6.5: Relative benefit of NOC over EOC, average distance, $\alpha = 1.0$

Note that like the results in Section 6.3.1, an increase in w results in more allocation of C among higher levels in the distribution tree. In addition, lower level nodes need a larger fraction of C to minimize objective function (6.9) for stronger temporal locality.

The story for the Dtelecom (Figure 6.9) and Level3 (Figure 6.10) topologies is quite different. A look at their features in Table 3.1 shows how they are different from Geant and Tiger. There are more edge nodes in these topologies; in addition, there are intermediate nodes with high degree in these two topologies.

For the Dtelecom topology, Figure 6.9a shows that a larger fraction of C is allocated to level 2 where the nodes with higher degree are located; although the fraction of C is distributed among edge nodes increases as temporal locality gets stronger. Figures 6.9b and 6.9c illustrate that the benefits of in-network caching is at least 25% and 45% for average retrieval distance and load on the server respectively. These two figures also show that the benefit of optimal in-network caching is constant over w. The reason is the allocation of a larger fraction of C to one level that has ICN nodes with high degree. Having $w = 0.5$ and $z = 10$ results in 36% shorter retrieval distance as well as 75% less load on the server in case of optimal in-network caching.

The same story is true for the Level3 topology, except that the node with higher degree is located at third level and its degree is much smaller than the degree of the similar node in Dtelecom topology. The relative benefit of in-network caching on average retrieval distance is at least 20%. This relative benefit is about 25%, 26% and 37% when z equals 1, 2 and 10 respectively. In-network caching also decreases the load



(a) $z = 1$ **(b)** $z = 2$

(c) $z = 10$

Figure 6.6: Relative benefit of NOC over EOC, average load on server, $\alpha = 1.0$

on the server by at least 35%. This benefit gets to 47%, 50% and 72$ when z equals 1,2 and 10.

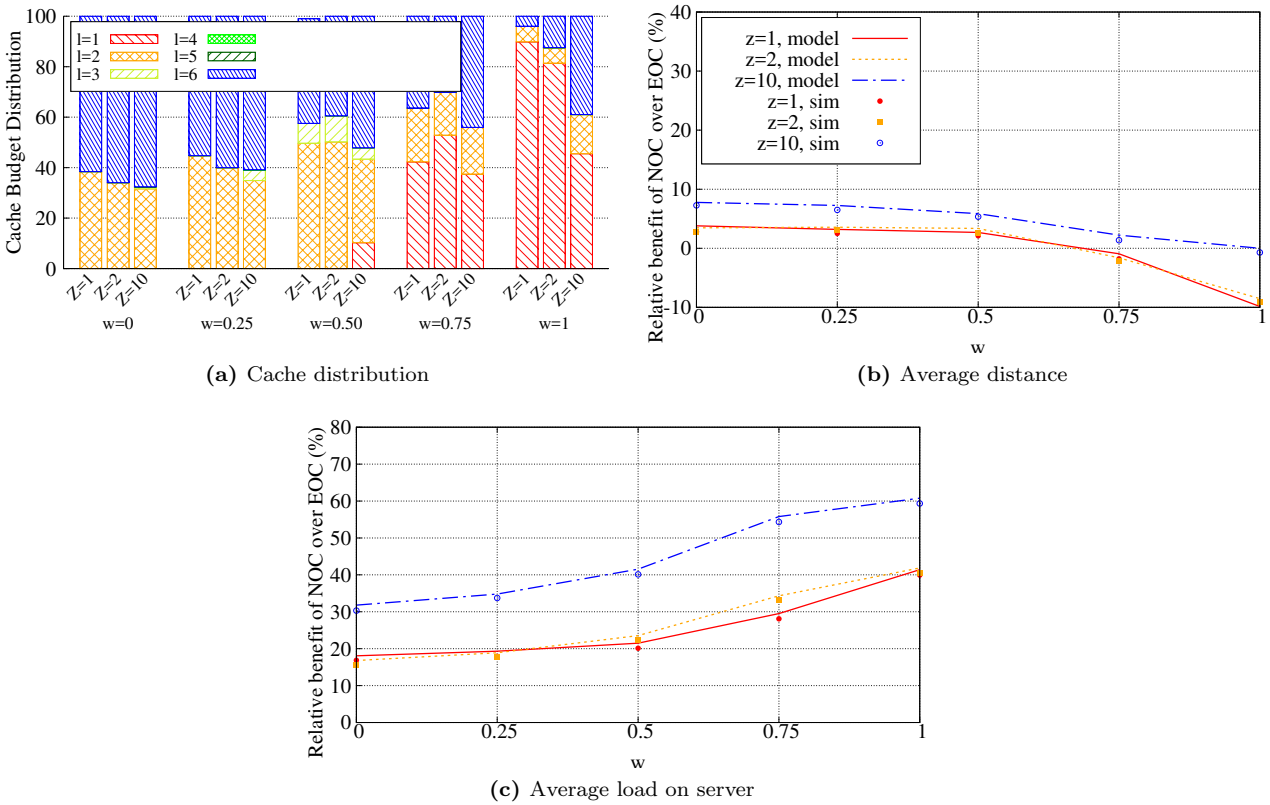
Findings

- The trend of optimal in-network distribution of the cache budget could be different for various topologies. Similar to the tree topology studied in Section 6.3.1, a larger fraction of C is allocated to intermediate ICN cache at higher levels in Geant and Tiger topologies as w increases. Contrary to Geant and Tiger, a large fraction of C is allocated to specific level regardless of w in the Dtelecom and Level3 topologies.

- Similar to the tree topology studied in Section 6.3.1, for data items with stronger temporal locality, optimization problem (6.9) distributes a larger fraction of C among levels closer to the edge.

6.4 Summary

This chapter modelled the distribution of users' requests in a non-IRM environment in the network as an optimization problem taking metrics from users' and ISP's point of view into account. The solution for this problem depicts an optimal distribution of cache budget C among the nodes in ICN networks. Studying



(a) Cache distribution

(b) Average distance

(c) Average load on server

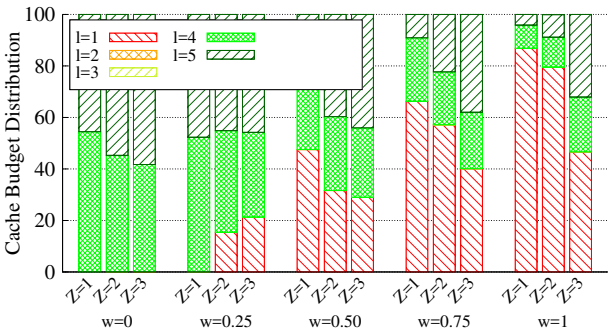
Figure 6.7: Optimal cache distribution, Geant, $\alpha = 1.0, C = 1000$

various settings for Zipf parameter for the distribution of users' requests (α), strength of temporal locality (z), total cache budgets (C) and topologies shows

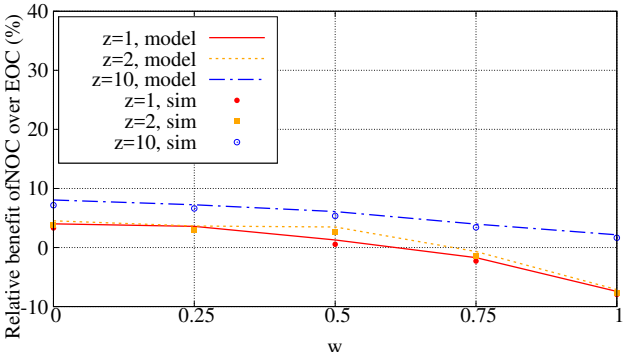
- Stronger temporal locality causes a larger distribution of total cache budget among edge nodes.
- As total cache budget of C decreases, fraction of C that is allocated to edge nodes shrinks.
- The efficiency of in-network caching on retrieval distance strongly depends on topologies and the strength temporal locality. For instance, the relative benefit of up to 10% is observed for $z = \{1, 2\}$ in tree, Geant and Tiger topologies. On the other hand, an optimal in-network caching in Level3 and Dtelecom topologies ends in at least 20% shorter retrieval distance.
- In-network caching is very helpful in decreasing the overall miss ratio for all settings. Less overall miss ratio in the system results in forwarding less traffic out of the local ICN networks (inter-network traffic) as well as smaller load on the servers.

The findings of this chapter show that in-network caching can be very effective in decreasing the overall miss ratio in ICN networks and relatively effective in shortening the retrieval distance that are more consistent with the studies of Danzig *et al.* [32] and Rossini *et al.* [79].

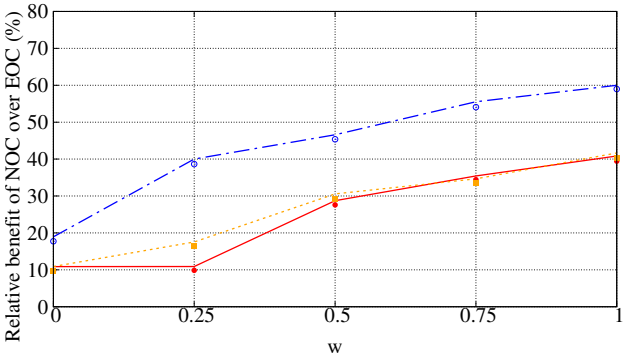
Note that the work in this chapter is being formatted to another conference paper, venue to be determined.



(a) Cache distribution

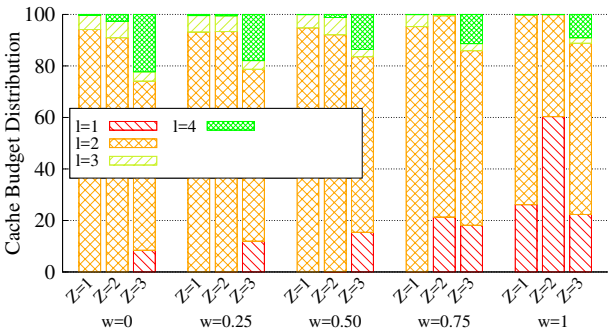


(b) Average distance

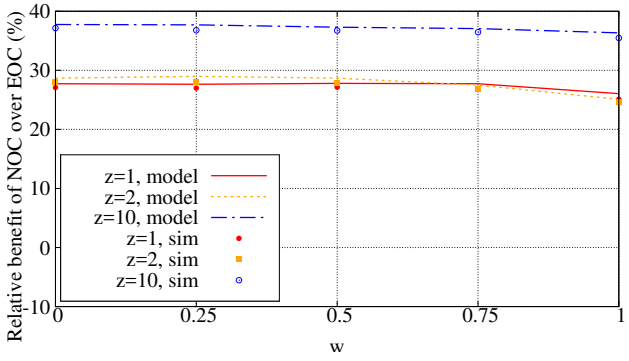


(c) Average load on server

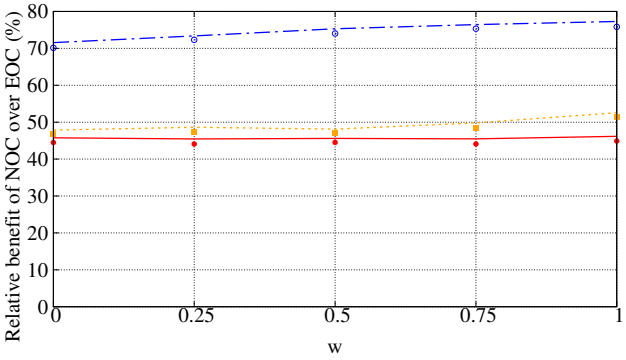
Figure 6.8: Optimal cache distribution, Tiger, $\alpha = 1.0, C = 1000$



(a) Cache distribution

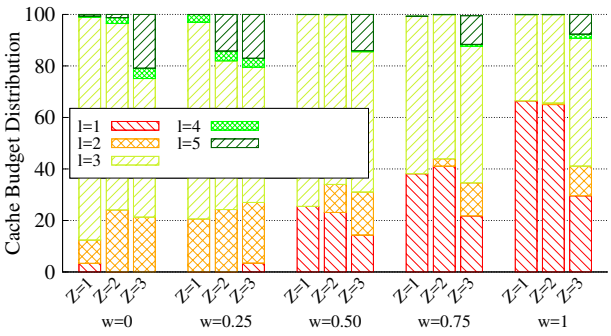


(b) Average distance

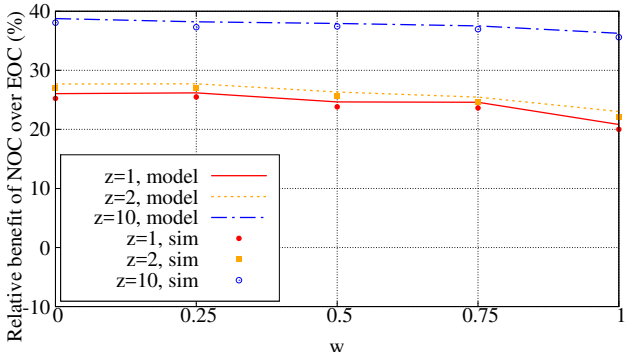


(c) Average load on server

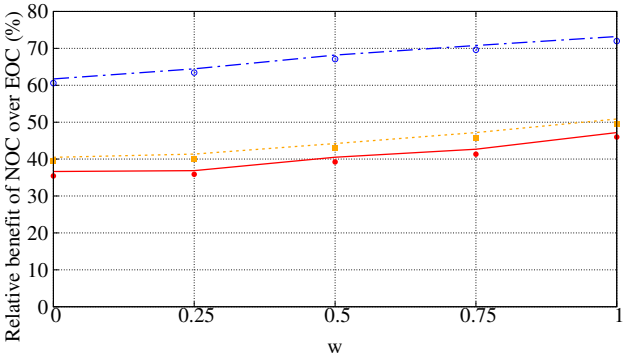
Figure 6.9: Optimal cache distribution, Dtelecom, $\alpha = 1.0, C = 1000$



(a) Cache distribution



(b) Average distance



(c) Average load on server

Figure 6.10: Optimal cache distribution, Level3, $\alpha = 1.0, C = 1000$

Chapter 7

Conclusion

7.1 Summary

This work is focused on modelling caching systems in realistic environments in which temporal and geographical locality are taken into account. Three major issues are addressed:

- First, a mathematical model for LRU-2 algorithm is proposed. The proposed model is evaluated against the simulation results. The simulation results show that the proposed model approximates the miss rate of LRU-2 algorithm accurately. The LRU-2 and 2-LRU cache replacement algorithms are also studied. The study shows that 2-LRU results in a higher and smaller hit ratio and miss rate respectively compared to LRU-2. In addition, the findings of this work also depicts that Gast's model for 2-LRU underestimates the miss rate for larger Zipf parameter (α) and cache sizes. Contrary to Gast's approximation for 2-LRU, the proposed model for LRU-2 calculates a better approximation of 2-LRU behaviour as either α or cache size increases. Our study also find that the accuracy of models' estimations at intermediate levels decreases due to the violation of IRM arrivals of requests at them.

- Second, an algorithm is proposed that generate users' requests in different regions following different Zipf distributions (i.e. the distribution of users' requests in regions have different arrival rate, different Zipf parameter and the order of popularity of items in regions are different) for different regions of the network while the overall distribution of users' requests in the network is still a Zipf distribution. The simulation results show that geographical locality causes different system behaviour in the simple test scenarios compared to identical request distribution. This work also illustrates that the gains brought by the distribution of cache budget among all the ICN nodes proportional to the rate of arriving requests at the nodes, while ignoring the distribution of users' requests and Zipf parameter α, is very small compared to the benefits obtained through edge caching.

- Finally, the efficiency of in-network caching is studied for LRU cache replacement and LCE replication algorithms in IRM and non-IRM environments. In this regard, the distribution of users' requests in ICN networks is modelled as an optimization problem. For non-IRM environment, the proposed Algorithm 5.4 and Garetto's model (2.14) are used to implement geographical and temporal localities respectively. Contrary to the second work, the distribution of user's requests arriving at ICN nodes is also taken into

account. The results show that in-network caching is effective in decreasing the miss ratio, load on the server and the distance to retrieve the data items.

7.2 Future Possible Research Areas

This section summarizes the assumptions considered as well as constraints encountered in this thesis. Relieving such assumptions alongside with resolving the constrains will be contributing to the possible future works.

The proposed LRU-2 model in Chapter 4 can be extended for users' requests with temporal locality. To this end, Garetto's model (Section 2.4) can be deployed to apply the popularity growth in users' requests. In addition, studying the performance of LRU-2 and 2-LRU under non-IRM environment will be interesting.

The study in Chapter 5 shows that investigating the influence of similar distributions for neighbouring regions, as the output of Algorithm 5.4, on local search is challenging. Modifying Algorithm 5.1 so that it creates similar regions through changing only the rank of data items while Zipf properties of the regions left unchanged could be a part of future work. In this case, studying the performance of similar distributions for users' requests on local search for neighbouring regions is more possible.

Investigating the optimization problem (6.9) for other cache replacement algorithms, such as 2-LRU, and other replication algorithms, such as LCD, will be interesting. As mentioned in Section 6.2, the optimization problem (6.9) is found difficult for cache replacement and replication algorithms other than LRU and LCE to get solved. Therefore, finding a mathematical justification for the failure of mathematical software like Matlab in solving such complicated optimization problems could be part of future works.

The optimization problem (6.9) also assumes an on-path caching strategy for ICNs. This optimization problem can be extended to support ICNs deploying off-path caching strategies [37, 59, 74]. The minimum miss ratio and retrieval distance found by corresponding optimization problem in an off-path caching mechanism then could be compared with the results found for on-path caching mechanism. Furthermore, extending (6.9) to support local search would be a good research study. Such optimization problem then can show how effective local search could be in minimizing the overall miss ratio in ICNs, load on the server and retrieval distance.

Another assumption considered in (6.9) is that all data items are hosted by one source node. This assumption can also be relieved so that several sources, which host permanent copies of data item, are available in the network. Investigating the performance of ICN caching mechanism with real-world traces [74] is also good research area as well.

In addition, finding practical solutions to implement optimal distributions of a cache budget among the ICN nodes would be a very good candidate as the next promising research area. Chu *et al.* for example, have deployed the utility optimization approach proposed by Dehghan *et al.* [34] to optimally allocate cached resources among content publishers [24].

References

[1] H. Abrahamsson and M. Nordmark. Program popularity and viewer behaviour in a large tv-on-demand system. In *Proceedings of the Internet Measurement Conference*, pages 199–210, Boston, MA, November 2012.

[2] L. A. Adamic and B. A. Huberman. Zipf's law and the Internet. *Glottometrics*, 3:143–150, 2002.

[3] J. M. Almeida, J. Krueger, D. L. Eager, and M. K. Vernon. Analysis of educational media server workloads. In *Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 21–30, Port Jefferson, NY, June 2001.

[4] C. Andersion. The long tail, 2004. http://arlt-lectures.com/The-Long-Tail.pdf/ [Online; accessed 12-June-2018].

[5] Z. Avramova, S. Wittevrongel, H. Bruneel, and D. D. Vleeschauwer. Analysis and modeling of video popularity evolution in various online video content systems: power-law versus exponential decay. In *Proceeding of the 1st International Conference on Evolving Internet*, pages 95–100, Cannes/La Bocca, France, August 2009.

[6] B. Baccala. Data-oriented networking. draft-baccala-data-networking-00.txt, IETF Internet Draft, August 2002.

[7] L. Backstrom, J. Kleinberg, R. Kumar, and J. Novak. Spatial variation in search engine queries. In *Proceedings of the 17th International Conference on World Wide Web*, pages 357–366, Beijing, China, April 2008.

[8] L. Backstrom, E. Sun, and C. Marlow. Find me if you can: improving geographical prediction with social and spatial proximity. In *Proceedings of the 19th International Conference on World Wide Web*, pages 61–70, Raleigh, NC, April 2010.

[9] D. S. Berger, P. Gland, S. Singla, and F. Ciucu. Exact analysis of TTL cache networks: the case of caching policies driven by stopping times. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems*, pages 595–596, Austin, TX, June 2014.

[10] C. Bernardini, T. Silverston, and O. Festor. Cache management strategy for CCN based on content popularity. In *Proceedings of the 7th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security: Emerging Management Mechanisms for the Future Internet*, pages 92–95, Barcelona, Spain, June 2013.

[11] C. Bernardini, T. Silverston, and O. Festor. A comparison of caching strategies for Content-Centric Networking. In *Proceedings of the IEEE Global Communications Conference*, pages 1–6, San Diego, CA, December 2015.

[12] Y. Borghol, S. Mitra, S. Ardon, N. Carlsson, D. Eager, and A. Mahanti. Characterizing and modelling popularity of user-generated videos. *Performance Evaluation*, 68(11):1037 – 1055, November 2011. Special Issue: Performance 2011.

[13] S. Borst, V. Gupta, and A. Walid. Distributed caching algorithms for content distribution networks. In *Proceedings of 29th Annual IEEE International Conference on Computer Communications and Networks*, pages 1478–1486, San Diego, CA, March 2010.

[14] J. Boyar, M. R. Ehmsen, and K.S. Larsen. Theoretical evidence for the superiority of LRU-2 over LRU for the paging problem. In *Approximation and Online Algorithms*, volume 4368 of *Lecture Notes in Computer Science*, pages 95–107. Springer Berlin Heidelberg, 2007.

[15] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino. Modeling data transfer in Content-centric Networking. In *Proceedings of the 23rd Internation Teletraffic Congress*, pages 111–118, San Francisco, CA, September 2011.

[16] M. Cha, H. Kwak, P. Rodriguez, Y. Ahn, and S. Moon. I Tube, You Tube, Everybody Tubes: analyzing the world's largest user generated content video system. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, pages 1–14, San Diego, CL, August 2007.

[17] M. Cha, H. Kwak, P. Rodriguez, Y. Ahn, and S. Moon. Analyzing the video popularity characteristics of large-scale user generated content systems. *IEEE/ACM Transactions on Networking*, 17(5):1357–1370, October 2009.

[18] W. K. Chai, D. He, I. Psaras, and G. Pavlou. Cache "less for more" in Information-Centric Networks. In *Proceedings of the 11th International IFIP TC 6 Conference on Networking*, pages 27–40, Prague, Czech Republic, May 2012.

[19] H. Che, Y. Tung, and Z. Wang. Hierarchical web caching systems: modeling, design and experimental results. *IEEE Journal on Selected Areas in Communications*, 20(7):1305–1314, September 2006.

[20] M. Chesire, A. Wolman, G. M. Voelker, and H. M. Levy. Measurement and analysis of a streaming-media workload. In *Proceedings of the 3rd Conference on USENIX Symposium on Internet Technologies and Systems*, pages 1–12, San Francisco, CL, March 2001.

[21] R. Chiocchetti, D. Rossi, and G. Rossini. ccnSim: An highly scalable CCN simulator. In *Proceeding of IEEE International Conference on Communications*, pages 2309–2314, Budapest, Hungary, June 2013.

[22] S. A. Chowdhury and D. Makaroff. Popularity growth patterns of YouTube videos - a category-based study. In *Proceedings of the 9th International Conference on Web Information Systems and Technologies*, pages 233–242, Aachen, Germany, May 2013.

[23] S. A. Chowdhury and D. Makaroff. Category-based YouTube request pattern characterization. *Web Information Systems and Technologies*, 189:154–169, January 2014.

[24] W. Chu, M. Dehghan, D. Towsley, and Z. Zhang. On allocating cache resources to content providers. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, pages 154–159, Kyoto, Japan, September 2016.

[25] CISCO. Cisco visual networking index (VNI) and VNI service adoption. Research report, CISCO, June 2016.

[26] E. G. Coffman, Jr. and P. J. Denning. *Operating Systems Theory*. Prentice Hall Professional Technical Reference, 1973.

[27] M. Gritte D. R. Cheriton. TRIAD: a scalable deployable NAT-based Internet architecture. Technical report, Stanford University, California, 2000.

[28] A. Dabirmoghaddam, M. Barijough, and J. Garcia-Luna-Aceves. Understanding optimal caching and opportunistic caching at "the edge" of Information-Centric Networks. In *Proceedings of the 1st ACM Conference on Information-Centric Networking*, pages 47–56, Paris, France, September 2014.

[29] J. Dai, Z. Hu, B. Li, J. Liu, and B. Li. Collaborative hierarchical caching with dynamic request routing for massive content distribution. In *Proceedings of the 31st Annual IEEE International Conference on Computer Communications*, pages 2444–2452, Orlando, FL, March 2012.

[30] A. Dan and D. Towsley. An approximate analysis of the LRU and FIFO buffer replacement schemes. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 143–152, Colorado, CO, 1990.

[31] C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren, and H. Karl. Network of information (NetInf) an Information-Centric Networking architecture. *Computer Communications*, 36(7):721 – 735, 2013.

[32] P. B. Danzig, R. S. Hall, and M. F. Schwartz. A case for caching file objects inside internetworks. In *Proceedings of ACM Conference on Communications Architectures, Protocols and Applications*, pages 239–248, San Francisco, CA, October 1993.

[33] M. Dehghan, B. Jiang, A. Dabirmoghaddam, and D. Towsley. On the analysis of caches with pending interest tables. In *Proceeding of the 2nd ACM Conference on Information-Centric Networking*, pages 69–78, San Francisco, CA, September 2015.

[34] M. Dehghan, L. Massoulie, D. Towsley, D. Menasche, and Y. C. Tay. A utility optimization approach to network cache design. In *Proceeding of the 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, San Francisco, CA, April 2016.

[35] V. Dimitrov and V. Koptchev. PSIRP project – Publish-Subscribe Internet Routing Paradigm: new ideas for future internet. In *Proceedings of the 11th International Conference on Computer Systems and Technologies*, pages 167–171, Sofia, Bulgaria, June 2010.

[36] A. B. Downey. Evidence for long-tailed distributions in the Internet. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 229–241, San Francisco, CA, November 2001.

[37] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K.C. Ng, V. Sekar, and S. Shenker. Less pain, most of the gain: incrementally deployable ICN. *SIGCOMM Computer Communication Review*, 43(4):147–158, August 2013.

[38] N. C. Fofack, P. Nain, G. Neglia, and D. Towsley. Analysis of TTL-based cache networks. In *Proceedings of the 6th International ICST Conference on Performance Evaluation Methodologies and Tools*, pages 1–10, Cargese, France, December 2012.

[39] N. C. Fofack, P. Nain, G. Neglia, and D. Towsley. Performance evaluation of hierarchical TTL-based cache networks. *Computer Networks*, 65(Supplement C):212 – 231, 2014.

[40] S. Foss, D. Korshunov, and S Zachary. Heavy-tailed and long-tailed distributions. In *An introduction to heavy-tailed and subexponential distribution*, chapter 2, pages 7–72. Springer Series in Operations Research and Financial Engineering, Springer, 2013.

[41] N. Fotiou, P. Nikander, D. Trossen, and G. C. Polyzos. Developing information networking further: from PSIRP to PURSUIT. In *Broadband Communications, Networks, and Systems*, pages 1–13, Athens, Greece, October 2010.

[42] C. Fricker, P. Robert, and J. Roberts. A versatile and accurate approximation for LRU cache performance. In *Proceeding of the 24th International Teletraffic Congress*, pages 1–8, Anaheim, CA, September 2012.

[43] C. Fricker, P. Robert, J. Roberts, and N. Sbihi. Impact of traffic mix on caching performance in a Content-Centric Network. In *Proceedings of the 31st Annual IEEE International Conference on Computer Communications*, pages 310–315, Orlando, FL, March 2012.

[44] M. Garetto, E. Leonardi, and V. Martina. A unified approach to the performance analysis of caching systems. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 1(3):12:1–12:28, May 2016.

[45] M. Garetto, E. Leonardi, and S. Traverso. Efficient analysis of caching strategies under dynamic content popularity. In *Proceedings of the 34th Annual IEEE International Conference on Computer Communications*, pages 2263–2271, Hong Kong, April 2015.

[46] N. Gast and B. Van Houdt. Asymptotically exact TTL-approximations of the cache replacement algorithms LRU(m) and h-LRU. In *Proceeding of the 28th International Teletraffic Congress*, volume 01, pages 157–165, Wrzburg, Germany, September 2016.

[47] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. Youtube traffic characterization: a view from the edge. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, pages 15–28, San Diego, CA, October 2007.

[48] S. Imai. *Design, Modelling, and Evaluation of Efficient Caching Mechanisms for Content Dissemination Networks*. PhD thesis, Osaka University, February 2015.

[49] V. Jacobsen, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. Braynard. Networking Named Content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 1–12, Rome, Italy, December 2009.

[50] Z. Jia, J. Huang, and C. Lin. Hierarchical caches in Content-Centric Networks: modeling and analysis. *Frontiers of Computer Science*, 9(6):846–859, December 2015.

[51] T. Johnson and D. Shasha. 2Q: A low overhead high performance buffer management replacement algorithm. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 439–450, Santiago de Chile, Chile, September 1994.

[52] J. Jung, A. W. Berger, and Hari Balakrishnan. Modeling TTL-based internet caches. In *Proceedings of the 22nd Annual IEEE International Conference on Computer Communications*, volume 1, pages 417–426 vol.1, San Francisco, CA, July 2003.

[53] K. Katsaros, G. Xylomenos, and G. C. Polyzos. Multicache: an overlay architecture for Information-Centric Networking. *Special Issue on Architectures and Protocols for the Future Internet Computer Networks*, 55(4):936 – 947, March 2011.

[54] Y. Kim and I.n Yeom. Performance analysis of in-network caching for Content-Centric Networking. *Computer Networks*, 57(13):2465 – 2482, 2013.

[55] T. Koponen, M. Chawla, B. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 181–192, Kyoto, Japan, August 2007.

[56] N. Laoutaris, H. Che, and I. Stavrakakis. The LCD interconnection of LRU caches and its analysis. *Performance Evaluation*, 63(7):609–634, July 2006.

[57] N. Laoutaris, S. Syntila, and I. Stavrakakis. Meta algorithms for hierarchical web caches. In *In Proceedings of IEEE International Conference on Performance, Computing, and Communications*, pages 445–452, Phoenix, AZ, April 2004.

[58] J. Li, B. Liu, and H. Wu. Energy-efficient in-network caching for Content-Centric Networking. *IEEE Communications Letters*, 17(4):797–800, April 2013.

[59] Y. Li, H. Xie, Y. Wen, and Z. Zhang. Coordinating in-network caching in Content-Centric Networks: model and analysis. In *Proceedings of the 33rd International Conference on Distributed Computing Systems*, pages 62–72, Washington, DC, July 2013.

[60] M. Lu, D. Sun, Y. Shi, and Y. Li. Peer-assisted streaming distribution over CCN. In *Proceedings of IEEE International Conference on Computer and Information Technology*, pages 444–451, December 2016.

[61] A. Mahanti, C. Williamson, N. Carlsson, M. Arlitt, and A. Mahanti. Characterizing the file hosting ecosystem: a view from the edge. *Performance Evaluation*, pages 1085 – 1102, November 2011.

[62] A. Mahanti, C. Williamson, and D. Eager. Traffic analysis of a web proxy caching hierarchy. *IEEE Network Magazine*, 14(3):16–23, May 2000.

[63] N. Megiddo and D. S. Modha. Outperforming LRU with an adaptive replacement cache algorithm. *Computer*, 37(4):58–65, April 2004.

[64] Z. Ming, M. Xu, and D. Wang. Age-based cooperative caching in Information-Centric Networking. In *Proceeding of the 23rd International Conference on Computer Communication and Networks*, pages 1–8, Shanghai, China, August 2014.

[65] A. Montazeri and D. Makaroff. Geographically-distinct request patterns for caching in Information-Centric Networks. In *Proceedings of the 42nd Conference on Local Computer Networks*, pages 579–582, Singapore, October 2017.

[66] L. Muscariello, G. Carofiglio, and M. Gallo. Bandwidth and storage sharing performance in Information-Centric Networking. In *Proceedings of the 1st ACM SIGCOMM Workshop on Information-Centric Networking*, pages 26–31, Toronto, Canada, August 2011.

[67] F. Olmos, B. Kauffmann, A. Simonian, and Y. Carlinet. Catalog dynamics: impact of content publishing and perishing on the performance of a LRU cache. In *Proceeding of the 26th International Teletraffic Congress*, pages 1–9, Karlskrona, Sweden, October 2014.

[68] E. J. O'Neil, P. E. O'Neil, and G. Weikum. The LRU-K page replacement algorithm for database disk buffering. *ACM SIGMOD Record*, 22(2):297–306, June 1993.

[69] H. Pinto, J. M. Almeida, and M. A. Gonçalves. Using early view patterns to predict the popularity of youtube videos. In *Proceedings of the 6th ACM International Conference on Web Search and Data Mining*, pages 365–374, Rome, Italy, February 2013.

[70] K. Poularakis and L. Tassiulas. On the complexity of optimal content placement in hierarchical caching networks. *IEEE Transactions on Communications*, 64(5):2092–2103, May 2016.

[71] I. Psaras, W. K. Chai, and G. Pavlou. Probabilistic in-network caching for Information-Centric Networks. In *Proceedings of the 2nd Edition of ACM SIGCOMM Workshop on Information-Centric Networking*, pages 55–60, Helsinki, Finland, August 2012.

[72] I. Psaras, R. G. Clegg, R. Landa, W. K. Chai, and G. Pavlou. Modelling and evaluation of CCN-caching trees. In *Proceedings of the 10th International IFIP TC 6 Conference on Networking*, pages 78–91, Valencia, Spain, May 2011.

[73] J. Ren, W. Qi, C. Westphal, J. Wang, K. Lu, S. Liu, and S. Wang. MAGIC: a distributed max-gain in-network caching strategy in Information-Centric Networks. In *Proceedings of IEEE Conference on Computer Communications Workshops*, pages 470–475, April 2014.

[74] A. Rizk, M. Zink, and R. Sitaraman. Model-based design and analysis of cache hierarchies. In *Proceedings of IFIP Networking Conference and Workshops*, pages 1–9, Stockholm, Sweden, June 2017.

[75] T. Rodrigues, F. Benevenuto, M. Cha, K. Gummadi, and V. Almeida. On word-of-mouth based discovery of the web. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement Conference*, pages 381–396, Berlin, Germany, November 2011.

[76] E. J. Rosensweig, J. Kurose, and D. Towsley. Approximate models for general cache networks. In *Proceedings of the 29th Annual IEEE International Conference on Computer Communications*, pages 1–9, San Diego, CA, March 2010.

[77] D. Rossi and G. Rossini. On sizing CCN content stores by exploiting topological information. In *Proceedings of 31st Annual IEEE International Conference on Computer Communications and Networks*, pages 280–285, Orlando, FL, March 2012.

[78] G. Rossini and D. Rossi. A dive into the caching performance of Content-Centric Networking. In *IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks*, pages 105–109, Barcelona, Spain, September 2012.

[79] G. Rossini and D. Rossi. Coupling caching and forwarding: benefits, analysis, and implementation. In *Proceedings of the 1st ACM Conference on Information-Centric Networking*, pages 127–136, Paris, France, September 2014.

[80] A. Rowstron and P. Druschel. Pastry: scalable, decentralized object location, and routing for large-scale Peer-to-Peer systems. In *IFIP/ACM Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, November 2001.

[81] L. Saino, I. Psaras, and G. Pavlou. Hash-routing schemes for Information-Centric Networking. In *Porceedings of thd 3rd ACM SIGCOMM Workshop on Information-Centric Networking*, pages 27–32, Hong Kong, China, August 2013.

[82] M. Saxena, U. Sharan, and S. Fahmy. Analyzing video services in web 2.0: a global perspective. In *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 39–44, Braunschweig, Germany, May 2008.

[83] S. Scellato, C. Mascolo, M. Musolesi, and J. Crowcroft. Track globally, deliver locally: improving content delivery networks by tracking geographic social cascades. In *Proceedings of the 20th International Conference on World Wide Web*, pages 457–466, Hyderabad, India, April 2011.

[84] M. Z. Shafiq, L. Ji, A. X. Liu, and J. Wang. Characterizing and modeling internet traffic dynamics of cellular devices. In *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, pages 305–316, San Jose, CA, June 2011.

[85] H. Shen, D. Wang, C. Song, and A. Barabási. Modeling and predicting popularity dynamics via reinforced Poisson processes. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 291–297, Quebec City, Canada, July 2014.

[86] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. In *Proceedings of Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 133–145, Pittsburgh, Pennsylvania, USA, August 2002.

[87] G. Szabo and B. A. Huberman. Predicting the popularity of online content. *Communications of the ACM*, 53(8):80–88, August 2010.

[88] R. Tewari, M. Dahlin, H.M. Vin, and J.S. Kay. Design considerations for distributed caching on the Internet. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, pages 273–284, Austin, TX, June 1999.

[89] S. Traverso, M. Ahmed, M. Garetto, P. Giaccone, E. Leonardi, and S. Niccolini. Temporal locality in today's content caching: why it matters and how to model it. *ACM SIGCOMM Computer Communication Review*, 43(5):5–12, November 2013.

[90] S. Traverso, M. Ahmed, M. Garetto, P. Giaccone, E. Leonardi, and S. Niccolini. Unravelling the impact of temporal and geographical locality in content caching systems. *IEEE Transactions on Multimedia*, 17(10):1839–1854, October 2015.

[91] G. Tyson, S. Kaune, S. Miles, Y. El-khatib, A. Mauthe, and A. Taweel. A trace-driven analysis of caching in Content-Centric Networks. In *Proceedings of the 31st Annual IEEE International Conference on Computer Communications*, pages 1–7, Munich, Germany, July 2012.

[92] G. Tyson, N. Sastry, R. Cuevas, I. Rimac, and A. Mauthe. A survey of mobility in Information-Centric Networks. *Communications of the ACM*, 56(12):90–98, December 2013.

[93] A. Varga and R. Hornig. An overview of the OMNeT++ simulation environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, pages 60:1–60:10, Marseille, France, March 2008.

[94] H. Wang, J. M. Hernandez, and P. Van Mieghem. Betweenness centrality in a weighted network. *Physical Review Edition*, 77:046105, April 2008.

[95] L. Wang, S. Bayhan, and J. Kangasharju. Optimal chunking and partial caching in Information-Centric Networks. *Computer Communications*, 61(C):48–57, May 2015.

[96] Y. Wang, Z. Li, G. Tyson, S. Uhlig, and G. Xie. Optimal caache allocation for Content-Centric Networking. In *Proceeding of the 21st IEEE International Conference on Network Protocols*, pages 1–10, Gotingen, Germany, October 2013.

[97] J. Wu, Y. Zhou, D. M. Chiu, and Z. Zhu. Modeling dynamics of online video popularity. *IEEE Transactions on Multimedia*, 18(9):1882–1895, September 2016.

[98] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, and G. C. Polyzos. A survey of Information-Centric Networking research. *IEEE Communications Surveys Tutorials*, 16(2):1024–1049, July 2013.

[99] Y. Zhu, M. Chen, and A. Nakao. CONIC: Content-Oriented Network with Indexed Caching. In *Proceedings of 29th Annual IEEE International Conference on Computer Communications and Networks*, pages 1–6, San Diego, CA, March 2010.

[100] M. Zink, K. Suh, Y. Gu, and J. Kurose. Watch global, cache local: YouTube network traffic at a campus network: measurements and implications. In *Proceedings of SPIE Multimedia Computing and Networking*, pages 681805–681805, San Jose, CA, January 2008.

Appendix A

Calculation Of S_1

Equation (4.5) is a differential-difference equation. To begin, define $g(x) = f(x + \frac{\tau}{2})$, so that the equation becomes

$$g'(x - \tau/2) = -\lambda_i e^{-\lambda_i(\tau - x)} g(\tau/2 - x). \tag{A.1}$$

Then with the change of variables $y = x - \tau/2$,

$$g'(y) = -\lambda_i e^{-\lambda_i(\tau/2 - y)} g(-y). \tag{A.2}$$

It is straightforward to see that any function of the form ae^{by} (for constants a and b) cannot be a solution. The next step should be to try a function of the form $a_1 e^{b_1 y} + a_2 e^{b_2 y}$ for constants $a_{1,2}$ and $b_{1,2}$. We guess that b_1 and b_2 are two roots of a quadratic, and take the form $b_{1,2} = \gamma \pm \beta$ for constants γ and β. Because there is only a first derivative on the left side, simply substituting $a_1 e^{b_1 y} + a_2 e^{b_2 y}$ will give a linear equation in b_1 and b_2. One way to get a quadratic in $b_{1,2}$ is to set $a_1 = \frac{1}{\sqrt{b_1}}$ and $a_2 = \pm\frac{1}{\sqrt{b_2}}$. So (after some trial and error), we settle on the general solution

$$g(y) = \frac{1}{\sqrt{\gamma + \beta}} e^{(\gamma+\beta)y} \pm \frac{1}{\sqrt{\gamma - \beta}} e^{(\gamma-\beta)y}. \tag{A.3}$$

We try the '$-$' solution first. Substituting into the equation,

$$\begin{aligned} 0 = {} & \frac{e^{(\lambda_i + 2\gamma)y - \lambda_i \frac{\tau}{2}}}{\sqrt{\gamma^2 - \beta^2}} \times \\ & \left(\sqrt{\gamma + \beta} e^{(\gamma+\beta)y} - \sqrt{\gamma - \beta} e^{(\gamma-\beta)y} \right) \\ & \times \left(\sqrt{\gamma^2 + \beta^2} e^{(2\gamma - \lambda_i)y + \lambda_i \frac{\tau}{2}} - \lambda_i \right). \end{aligned} \tag{A.4}$$

The first term cannot be 0. The middle factor has no dependence on λ_i, so we move to the third factor. γ and β must be independent of y, and this can be achieved by setting $\gamma = \lambda_i/2$. The third factor then reduces to

$$\sqrt{(\lambda_i/2)^2 - \beta^2} e^{\lambda_i \tau/2} - \lambda_i, \tag{A.5}$$

which has roots

$$\beta = \pm\frac{\lambda_i}{2}\sqrt{1 - 4e^{-\lambda_i \tau}}. \tag{A.6}$$

It does not matter which one we pick, so take the '$+$' root. Note that if we had taken the '$+$' solution to $g(y)$ then we would not have been able to solve the resulting equation, so we made the right choice. The general solution is

$$g(y) = \frac{1}{\sqrt{\mu_1}} e^{\mu_1 y} - \frac{1}{\sqrt{\mu_2}} e^{\mu_2 y}, \tag{A.7}$$

with

$$\mu_{1,2} = \frac{\lambda_i}{2}(1 \pm \sqrt{1 - 4e^{-\lambda_i \tau}}). \tag{A.8}$$

Thus

$$f(x) = g(x - \tau/2) = \frac{1}{\sqrt{\mu_1}} e^{\mu_1 (x - \frac{\tau}{2})} - \frac{1}{\sqrt{\mu_2}} e^{\mu_2 (x - \frac{\tau}{2})}. \tag{A.9}$$

The solution we want is $\mathbb{E}[S_1(x)] = Rf(x)$ for some constant R. We determine the appropriate value of R for our situation via the boundary condition $\mathbb{E}[S_1(\tau)] = \frac{1}{\lambda_i}$, implying

$$R = \frac{1}{\lambda_i}\left(\frac{1}{\sqrt{\mu_1}} e^{\mu_1 \frac{\tau}{2}} - \frac{1}{\sqrt{\mu_2}} e^{\mu_2 \frac{\tau}{2}} \right)^{-1}. \tag{A.10}$$

Appendix B

Calculation Of S_2

For now we continue to assume that $x < \tau$. Equation (4.11) is almost the same as (4.5), with the only difference being the lack of a minus sign on the right side. Unsurprisingly, the same ideas as before all work. The only difference is at (A.3), where we take the '+' solution instead of the '−' one. The general solution is thus

$$f(x) = \frac{1}{\sqrt{\mu_1}} e^{\mu_1 (x - \frac{\tau}{2})} + \frac{1}{\sqrt{\mu_2}} e^{\mu_2 (x - \frac{\tau}{2})}, \tag{B.1}$$

with $\mathbb{E}[S_2(x)] = Rf(x)$ for some R. Taking $x \to \tau$ gives $R = \frac{\mathbb{E}[S_2(x)]}{f(\tau)}$ with

$$f(\tau) = \frac{1}{\sqrt{\mu_1}} e^{\mu_1 (\frac{\tau}{2})} + \frac{1}{\sqrt{\mu_2}} e^{\mu_2 (\frac{\tau}{2})} \tag{B.2}$$

as per (B.1), and with $\mathbb{E}[S_2(x)]$ still unknown. On the other hand, taking $x \to \tau$ in (4.10) gives

$$\mathbb{E}[S_2(x)] = \frac{1}{\lambda_i} + \int_0^\infty \lambda_i e^{-\lambda_i u} \mathbb{E}[S(u)] du. \tag{B.3}$$

Now consider the case that $x > \tau$. It is impossible to terminate after u_1', so in this case we are forced to sample (at least) u_2'. Thus the time at which we terminate is independent of x, and hence so too is $S_2(x)$. Thus $\mathbb{E}[S_2(x)]$ is a constant, and we must have $\mathbb{E}[S_2(x)] = \mathbb{E}[S_2(\tau)]$ for $x \geq \tau$. Splitting the integral of (B.3) into two parts,

$$\begin{aligned}
\mathbb{E}[S_2(x)] &= \int_0^\tau \lambda_i e^{-\lambda_i u} \mathbb{E}[S_2(u)] du \\
&\quad + \int_\tau^\infty \lambda_i e^{-\lambda_i u} \mathbb{E}[S_2(u)] du + \frac{1}{\lambda_i} \\
&= \int_0^\tau \lambda_i e^{-\lambda_i u} \frac{\mathbb{E}[S_2(\tau)]}{f(\tau)} f(u) du \\
&\quad + \int_\tau^\infty \lambda_i e^{-\lambda_i u} \mathbb{E}[S_2(\tau)] du + \frac{1}{\lambda_i} \\
&= \frac{1}{\lambda_i} + Q\mathbb{E}[S_2(\tau)] + e^{-\lambda_i \tau} \mathbb{E}[S_2(\tau)]
\end{aligned} \tag{B.4}$$

where

$$\begin{aligned}
Q = \int_0^\tau \lambda_i e^{-\lambda_i u} \frac{f(u)}{f(\tau)} du &= \frac{2}{\sqrt{\mu_1} e^{\mu_2 \frac{\tau}{2}} + \sqrt{\mu_2} e^{\mu_1 \frac{\tau}{2}}} \\
\times \left(\sqrt{\mu_1} \sinh\left(\mu_2 \frac{\tau}{2}\right) + \sqrt{\mu_2} \sinh\left(\mu_1 \frac{\tau}{2}\right) \right).
\end{aligned} \tag{B.5}$$

Simplifying (B.4) results in (4.12).

Appendix C

ccnSim Configurations

Code C.1: node.ned in ccnSim.

```
package modules.node;
import modules.node.core.*;
import modules.node.strategy.*;
import modules.node.cache.*;
import modules.node.Inode;


module node like Inode{
    parameters:
        string CL = default("core_layer");
        string RS = default("lru_cache");
        string FS = default("spr");
        content_store.DS = default("lce");

    gates:
        inout face[];
        inout client_port;

    submodules:
        core_layer     : < "modules.node.core." + CL > like core;
        strategy_layer : < "modules.node.strategy." + FS > like strategy;
        content_store  : < "modules.node.cache." + RS > like cache;

    connections :
        for i=0..sizeof(face)-1{
            core_layer.face++ <--> face [i];
        }

        core_layer.client_port    <--> client_port;
        core_layer.cache_port     <--> content_store.cache_port;
        core_layer.strategy_port  <--> strategy_layer.strategy_port;
}
```

Code C.2: Topology settings in omnetpp.ned.

```
#General paramteters
[General]
network = networks.${net=dtelecom}_network
######################## Repositories ##############################
**.node_repos = ""
**.num_repos  = ${numRepos = 1 }
```

Code C.3: Clients settings in omnetpp.ned.

```
######################### Clients ###################################
**.node_clients = ""
**.num_clients = ${numClients = 1 }
**.lambda = ${lam = 4 }
## Indicates the type of the simulated clients: IRM, ShotNoise, GeographicalLocality,
    ls_IRM and ls_GeographicalLocality
**.client_type = "client_${clientType = IRM }"
```

Code C.4: Local search settings in omnetpp.ned.

```
**.CL = "${ cl = core}_layer"
**.localsearch_timeout = 0.05
## local_search_depth should be at least 1
**.local_search_depth = ${lsd=1}
```

Code C.5: Content distribution settings in omnetpp.ned.

```
**.file_size = 1
##Shaping factor of the Zipf distribution
**.alpha = ${alp=1.4}
**.objects = ${totCont = 1e5}
## Content distribution type: IRM, ShotNoiseContentDistribution,
    GeographicalLocalityContentDistribution
**.content_distribution_type = "${contDistrType =  IRM  }"

############ Geographical Locality Content Distribution ############
# Configuration File for the  Geographical Locality Model
**.request_patterns_file = "input_files/rp/${RPF=rp_dtel_edges1}.dat"
**.client_to_region_map_file = "input_files/ntrm/${CTRM=ntrm_dtel_edges1}.dat"
```

Code C.6: Forwarding settings in omnetpp.ned.

```
**.FS = "${ fs = spr }"

#nrr parameters
**.TTL2 = ${ttl = 1000}
**.TTL1= ${ttl}
**.routing_file = ""
```

Code C.7: Caching settings in omnetpp.ned.

```
**.DS = "${ mc = lce }"
**.RS = "${ rs = k2lru }_cache"
##Cache size (in chunks)
**.cache_budget_file = "input_files/cb/${CBF = cb_dtelecom_100}.dat"
## for k2lru, how larger the virutal cache is than the physical cache?
**.vc_size_multiple_factor = ${VCMPC = 1}
# q is used only in qlru/qk2lru caching algorithm
**.q = ${Q=0.01}
```